NBSIR 87-3624

The National Bureau of Standards Programmers Guide for the Field Materiel-Handling Robot (FMR)

S. Szabo

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Manufacturing Engineering
Robot Systems Division
Gaithersburg, Maryland 20899

September 1987

Sponsored by:
United States Army
Human Engineering Laboratory
Aberdeen Proving Ground, Maryland 21005-5001

NBSIR 87-3624

THE NATIONAL BUREAU OF STANDARDS PROGRAMMERS GUIDE FOR THE FIELD MATERIEL-HANDLING ROBOT (FMR)

S. Szabo

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Manufacturing Engineering
Robot Systems Division
Gaithersburg, Maryland 20899

September 1987

Sponsored by:
United States Army
Human Engineering Laboratory
Aberdeen Proving Ground, Maryland 21005-5001

U.S. DEPARTMENT OF COMMERCE, Clarence J. Brown, ***Acting Secretary***

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, ***Director***

October 4, 1987

APPROACH-PALLET
PICKUP-PALLET

1. Introduction

The National Bureau of Standards is contracted by the United States Army's Human Engineering Laboratory (HEL) to develop a high level sensory interactive robot controller based on the NBS Real-Time Control System (RCS). The controller will be incorporated within the HEL Field Material-Handling Robot (FMR). The primary task of the FMR is the handling of palletized loads; the robot is equivalent to a sensor guided fork lift.

The basic RCS configuration is extensively documented in the NBS Real-Time Control System User's Reference Manual. The following document is the supplement to the RCS Reference Manual. It describes the extensions to RCS that are specific to the FMR. It is assumed that the audience has read the RCS Manual before he or she attempts to read this manual. In any case references to the RCS Manual are given and the RCS Manual table of contents and index should also provide some assistance.

The Robot Sensor Language (RSL), developed at NBS for use with the FMR Real-Time Control System, features sensory interactive off-line (pre-planned) programming. An in depth description of RSL is also presented in the RCS Reference Manual. Several Path Point commands were developed to make use of the FMR sensor package and to program the type of tasks NBS envisioned the FMR would be expected to perform.

Four sections are covered in this document: the sensor modelling techniques used in conjunction with the NBS sensor package, the FMR Path Point commands, two RSL programs used for transferring 155mm pallets, and the real-time PATH level code that executes the Path Point commands.

1.1. Documentation Conventions

This document uses bold face text to distinguish between program (software) statements and text. Program statements include names of routines, variables and commands that can be entered from a terminal into the RCS. One exception is the bold face used to highlight section numbers and titles.

The syntax for the path point commands in Section 2 use the vertical bar to separate parameter fields.

The FORTH programming language supports some documentation conventions; primarily the left and right parenthesis to bound comments and the percent sign to indicate the remaining portion of the line is a comment.

2. Sensor Modelling

The NBS Real-Time Control System for the FMR utilizes two types of sensors: Acoustic sensors (ultrasonic sonars) provide range data while optical emitter/detector pairs provide proximity information. (Detail information on the sensors can be found in the NBS GFE Technical Data Package for the FMR May 1986.) Two modelling schemes were developed which allow the RCS controller to effectively use the sensors. One scheme models each of the ultrasonic ranging devices. The second scheme models the relationships between the proximity detector configurations and the pallets. The following sections describe the modelling and calibration techniques for each sensor.

2.1. Sonar Modelling

The ultrasonic sonars used in the FMR sensor package have been characterized through a process of benchmark testing and live usage. The results of the characterization are embedded in a model used by the control system. The model is stored as a data structure within the RCS file system. The primary benefit of such a scheme is the separation of data from control procedures. Previously the sonar characteristics were stored inside the control level or were entered by the user as parameters within a Path Point command. Any changes to the sonars required modification of the control level or having the user perform some calculation and entering the results through the command call. With the sonar model, any changes only require modification of the model represented in the data structure. The PATH control level interprets the model for data necessary in the execution of control functions such as sensor guided translations and rotations.

2.1.1. Model Data for the Sonars

The model for each sonar consists of the following data stored as fields in a RCS record structure.

snr-name	The name of the sonar; one of the names contained in the variable owner **snr-names**.
snr-id	An identification number, 0 thru 15.
snr-type	The sonar type; currently Polaroid (0) and Migatron (1) are supported.
snr-axis	The axis of the tool point frame (X,Y,Z) which is parallel to the sonar line of sight axis.
snr-mtb	Name of a sonar movetable previously defined which gives the position of a sonar relative to the tool frame. The movetable does not require any rotations since currently only the position of the transducer is used.
snr-angle	The sonar typically generates a field pattern similar to a cone. This record field holds the full angle of the cone in degrees.
snr-min-range	The minimum effective range of the sonar.
snr-max-range	The maximum effective range of the sonar.

2.1.2. Compiling the Sonar Model

The RSL level word **-snr-** compiles the model. The first step is to edit a block from the RSL level to include all the parameters associated with a sonar. For example:

-snr- snr-0
0
Polaroid
X
snr0-mtb
10.0
12.5
128.0

The next step is to load the block which compiles the parameters into the following record and file:

```
50 bytes VAR-O snr-model-para
6 strv  "snr-name"
iv      snr-id
iv      snr-type
iv      snr-axis
iv      snr-mtb-^
fv      snr-angle
fv      snr-min-range
fv      snr-max-range
20 rec FILE  SONAR-MODEL-FILE
```

2.1.3. Interpreting the Sonar Model

Once the model is compiled into the RCS file system, the PATH level can interpret the **SONAR-MODEL-FILE** file. To facilitate retrieving the record of a sonar the PATH level maintains an array of pointers, **snr-model-array**, which maps a physical sonar number (0 thru 15) into a record in the file **SONAR-MODEL-FILE.** The routine **INIT-SNR-MODEL-ARRAY** maintains the array and is executed at power up. An element in **snr-model-array** that contains a 0# indicates the sonar does not exist. The user can retrieve the record of a sonar from within a routine in the following manner:

```
SONAR-MODEL-FILE  % set file
snr-model-array { sonar# } => record#  % set record# with desired pointer
retrieve    % retrieve model record
```

An example of how to use the sonar model can be found in the EQUATE Path Point command routines located on the PATH level (section 5.7.8.3. **EQUATE-SONAR-MODEL-INIT**).

2.1.4. Calibrating the Sonar Model

The only calibration required for the sonar model is the position of the transducer. A movetable is used to define the position of the sonar relative to the tool coordinate frame. The location of the tool frame is specified by the wrist frame and an added tool movetable called **TOOL-MTB**. The tool frame is specified by the programmer and typically only changes with the end-effector. If the tool frame is changed, then an appropriate change must also be made in the sonar movetables.

2.2. Proximity Detector Modelling

The proximity detectors for the FMR are used to prevent the fork tines from colliding with the pallet support structures (feet) during insertion. Eight detectors are positioned at the tip of FMR fork tines (see Figure 2.1.). Figure 2.1. shows the relationship between the fork and a 155mm pallet support structure prior to insertion (perspective view). The top view shows the desired alignment between the fork and the structure. An enlargement of that view illustrates the relationship between the left

tine with it's proximity detectors and the pallet foot (right tine is a mirror image with detectors 4-7). Note that the proximity detector's field of view is very narrow (1-3 degrees) and the sensitivity of the detector allows them to pickup obstacles at approximately 4-6 inches (this is adjustable).

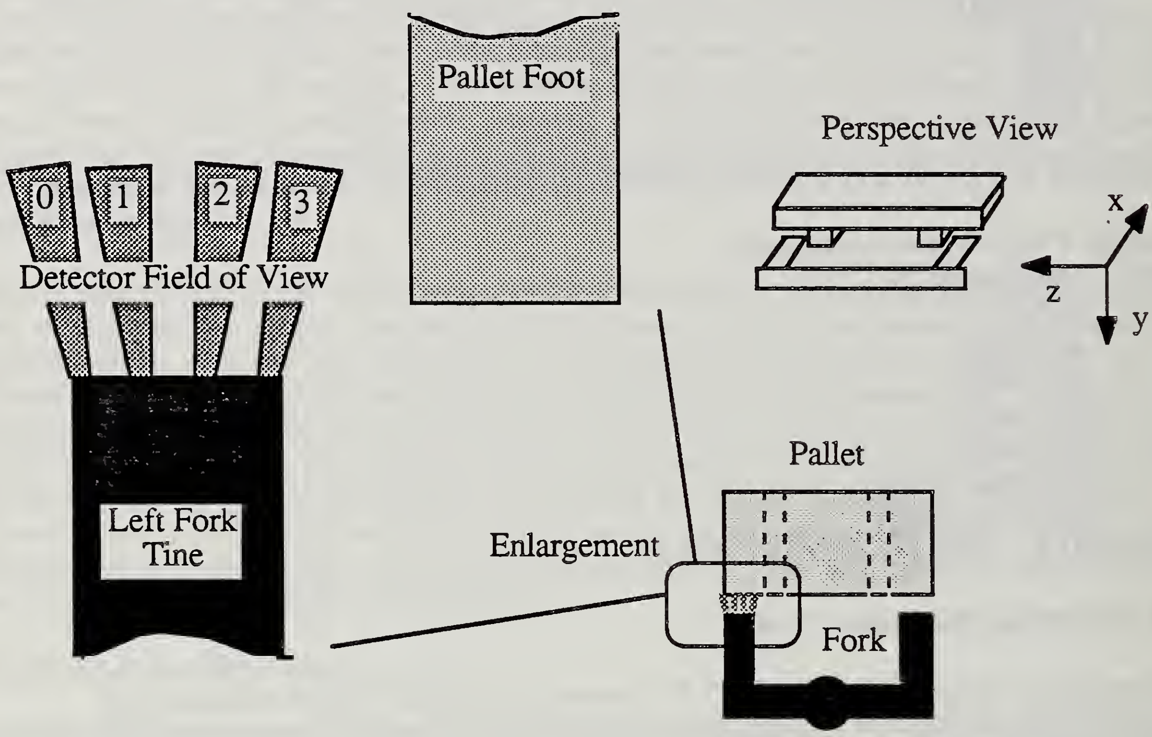


Figure 2.1. The relationship between the fork proximity detectors and a 155mm pallet.

The proximity detector configuration allows the FMR to correct a misalignment (translation along the Z axis of the tool frame) of approximately plus/minus six inches from the ideal insertion position. Figure 2.2 shows an example of how the proximity sensors 1, 2 and 3 detect a misalignment in the -Z direction.

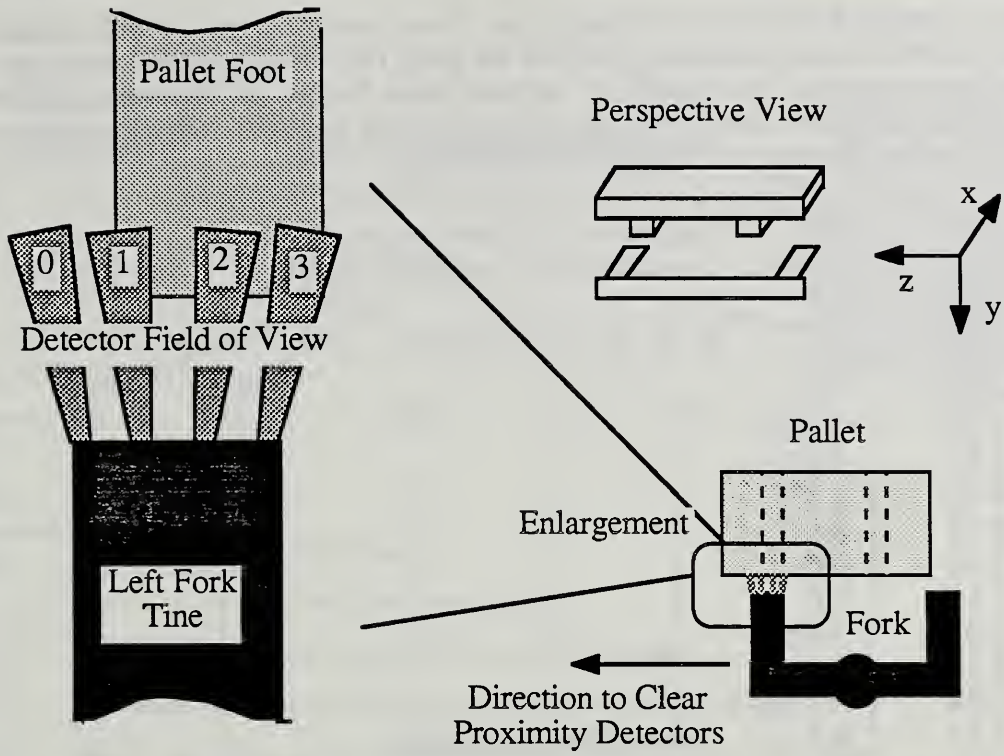


Figure 2.2. The Proximity sensors detecting a misalignment between the fork and pallet.

Detector 0 limits the worst case misalignment (6 inches) that the FMR can correct in the -Z direction. The Path Point command **pickup-pallet** uses the proximity detector configuration to correct misalignments (sections 3.7. and 5.7.12.).

The modelling for the detectors uses a simple technique which optimizes run-time execution by compressing as much information as possible into the model in an off-line fashion. The technique is to compile a look-up table which directly maps an active detector into the necessary motion to clear the obstacle. For the case of the 155mm pallets the look-up table is generated by placing the pallet in front of the fork tines to block a particular combination of detectors. The pallet is then moved until the detectors are clear from the foot. Finally the movement is measured and stored into the look-up table. This is repeated for all combinations of detector blockages. At run-time the detectors are read and the combination of the active detectors are used to directly access the lookup table and retrieve the required movement to clear the blockage.

2.2.1. Model Data for the Proximity Detectors

Two floating point arrays are used to store the look-up tables for the left and right tine proximity detector configurations: **l-tine-calib-table** and **r-tine-calib-table**. Each element in the array contains either a value indicating the translation motion for the associated active detectors or an error. Currently the error value is the number 0.0 and the error condition indicates an impossible combination of active detectors. As an example, in the case of the 155mm pallet it is impossible for detectors 1 and 3 (see Figure 2.2.) to be active while detector 2 is not. If this does occur it indicates one of the detectors are bad, misaligned or possibly obstructed by dirt. Another possibility is the pallet foot does not measure up to expectations.

2.2.2. Compiling the Proximity Detector Model

The two tine calibration tables are edited into a power-up block on the PATH level. The blocks are

loaded each time the PATH level is brought up. If any modification is made to the table, the block must be reloaded. It is not necessary to save the table (ie. **MEM>DISK**) since the table is always loaded during power-up. An example of the load block for the left tine table follows. The comments (within parentheses) serve two purposes: first they show the array element number and second they illustrate the active detectors associated with the entry.

```
( 0 0 0 0 )    0.0
( 0 0 0 1 )    8.25
( 0 0 1 0 )    6.0
( 0 0 1 1 )    7.0
( 0 1 0 0 )    4.0
( 0 1 0 1 )    0.0
( 0 1 1 0 )    4.875
( 0 1 1 1 )    5.75
( 1 0 0 0 )    1.875
( 1 0 0 1 )    0.0
( 1 0 1 0 )    0.0
( 1 0 1 1 )    0.0
( 1 1 0 0 )    3.625
( 1 1 0 1 )    0.0
( 1 1 1 0 )    3.875
( 1 1 1 1 )    0.0      ==> l-tine-calib-table
```

2.2.3. Interpreting the Proximity Detector Model

The Path Point command **pickup-pallet** is an example of how the proximity detector model is interpreted. The NBS system currently only supports the model for the 155mm pallet. The system can easily be extended to support a model for each different type of pallet. This would require the development of a model compiler word similar to the sonar model compiler **-snr-**.

2.2.4. Calibrating the Proximity Detector Model

The look-up table is generated by positioning the pallet in front of each proximity sensor and moving the pallet until it no longer detects the foot. This measures the field pattern, the position and the orientation of the sensor (with relation to the pallet foot) while also measuring the reflective properties of the pallet surface. The entry into the table is the measurement of the distance the pallet is moved. A plus or minus sign is included to indicate the direction the fork needs to travel to clear the foot. This process must be repeated for each combination of active detectors. The remaining combinations of active detectors that cannot be achieved by positioning the pallet are flagged with a 0.0 indicating a possible error condition.

Achieving each combination of sensors is not difficult. It is recommended that a pallet be placed on wheels so that it can be slowly passed by the detector configuration until there is a change in the combination of active sensors. The change can be seen by viewing the LED indicators on each proximity board in the interface chassis or by using the **DISPLAY-PROXIMITY-SWITCHES** routine (section 5.9.2.). A sheet of paper taped to the floor is used to mark each change in the detector combination and the position where the change occurred. Once complete the paper is removed and measurements are made from the point on the paper where each combination of detectors were active to the common point where all detectors were inactive. Remember that the measurement must reflect the motion and direction the fork needs to travel to guarantee the detectors become inactive. Figure 2.3 shows a sample sheet generated during the calibration of the left tine proximity detector configuration with a 155mm pallet foot. Each occurrence of a change in active detector combinations is recorded while the pallet is moved from left to right. The active detectors are shown starting with a 0000 to indicate all detectors are inactive. Detector 0 is the first to see the pallet

and detector 3 is the last.

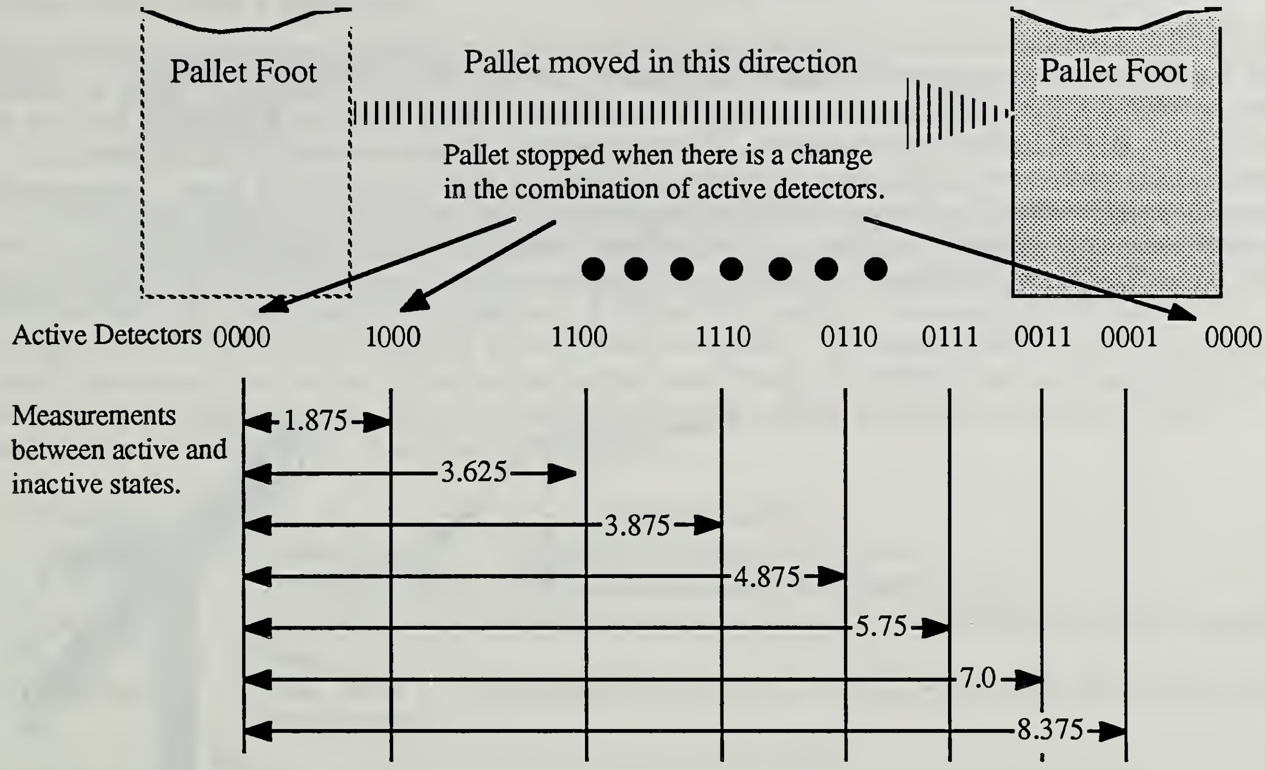


Figure 2.3. Example calibration of left fork tine proximity detector configuration.

3. Path Point Commands for the FMR

The FMR Path Point commands give the programmer the capability to position and orient the FMR fork end-effector using sensor feedback supplied by the FMR sensor package. Figure 3.1 shows a fork mockup with the NBS sensor package. A coordinate frame is shown attached to the fork and is useful for specifying tool based motions. For the purpose of this document, it is sufficient to visualize the position of the frame as between and parallel with the tips of the tines. The package currently consists of eight Polaroid and five Migatron ranging devices, and eight optical emitter/detector proximity switches (the NBS FMR Technical Data Package, Section 5.6.1. **SONAR-READ** and Section 2.Sensor Modelling give further information on the sensor package).

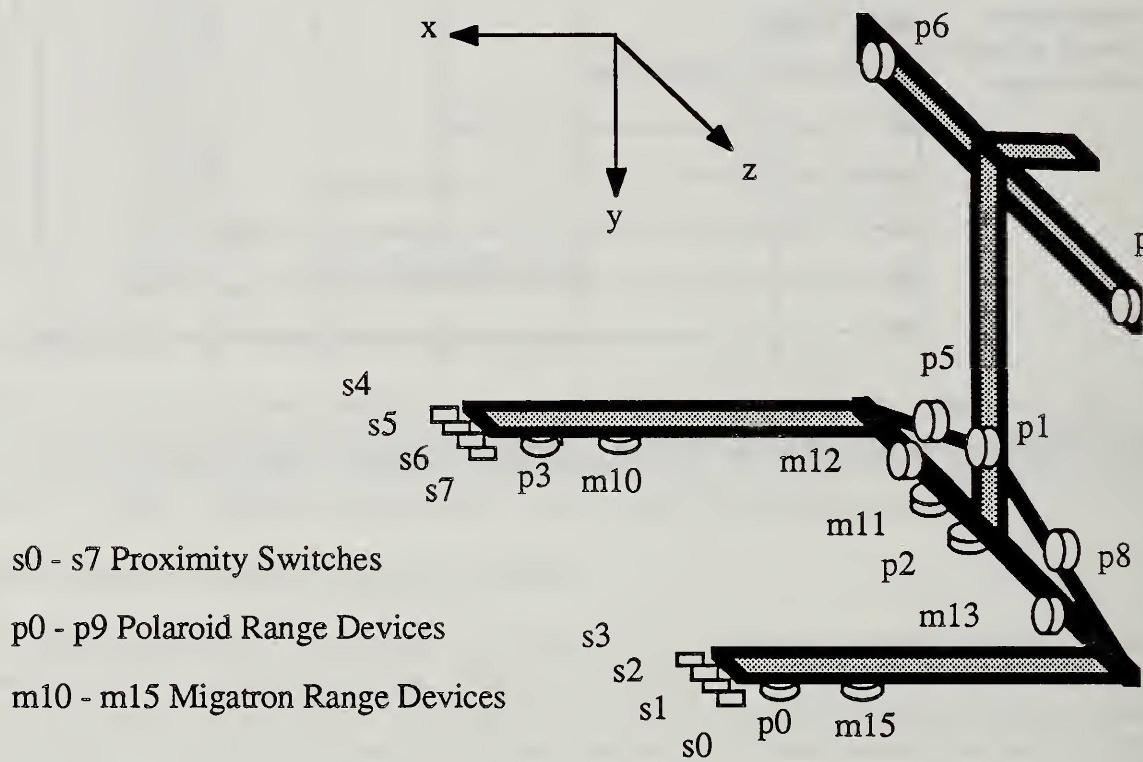


Figure 3.1. The FMR Sensor Package with Tool Coordinate Frame

The Path Point commands can be combined in a logical sequence to perform such tasks as aligning the fork with a truck bed, locating a randomly oriented pallet and transferring an array of pallets. Briefly the **range** and **edge** commands provide translation and pallet recognition capabilities; **equate** allows the fork to be rotated and aligned; **scan, align-grip, approach-pallet**, and **pickup-pallet** are specialized commands for locating a 155mm pallet, determining the pallet entry side, aligning and preparing the fork for insertion, and inserting the tines respectively. Some additional commands are also included. A description of the Path Point commands and their syntax is discussed next.

The parameters for a Path Point command vary but two types of parameters are supported by the base RSL system: the Trajectory Phrase and the Location Phrase. In this section, the words **traj phrase** and **loc phrase** are used for the Trajectory Phrase and the Location Phrase respectively. Their specific format is explained in the RCS Manual (Appendix B-70). The reader can look ahead to Section 4 for examples of the usage of Path Point commands in the context of an RSL program. The

specific commands are now examined.

3.1. Range Path Point Command

range | **r-sonar** | **r-range** | **r-threshold** | **r-axis** | **halt-enable** |
| **loc phrase** |
| **traj phrase** |

The **range** command is used to position the fork a specified range from a target. The parameters specify a move along the **r-axis** axis of the cartesian frame specified by the **loc phrase** (only **tool nul** frame is supported) to the range measured by the sonar **r-sonar#**, within **r-threshold.** The commanded translation is positive if **r-range** - **r-sonar#**{range-value} > 0 ({range-value} is the range returned by the sonar **r-sonar#**). A **sonar-limit** error is returned if a range is reported below that of the the minimum sonar range detectable (the minimum is derived from the model of the sonar). This can occur if the wrong sonar number is entered or if the wrong sonar is selected (ie. Polaroid vs. Migatron).

r-sonar# { **0 - 15** } - Sonar to use at execution time.
r-range { floating point } - Desired sonar range to achieve.
r-threshold { floating point } - Acceptable threshold for desired range.
r-axis { **X, Y, Z** } - The axis of translation; one of the cartesian axes of the location phrase.
halt-enable { **true, false** } - The robot will come to a complete halt after this command if the flag is true.

3.2. Edge Path Point Command

edge | **e-sonar#** | **e-range** | **e-delta** | **halt-enable** |
| **loc phrase** |
| **traj phrase** |

The **edge** command is used to locate edges. An edge is defined as a difference in range readings of at least **e-delta.** The set of range readings bounded by **e-delta** must also include the value specified in **e-range.** In other words **e-delta** describes the magnitude of the edge and **e-range** declares at what distance to search for it (see Section 5.7.7.4. **EDGE**). If the **halt-enable** flag is true the system initiates a halt when the sonars first satisfy the edge conditions. The command does not servo to the sensor condition. If the sonar condition is not satisfied after the halt is completed (due to overshoot, for example), the error **sensor-cond** is reported. If the robot reaches the location before the sonar condition is satisfied the error **point-reached** is reported. The **loc phrase** is the robot goal and is used to specify where to search for the edge. The **traj phrase** is used to specify the type of motion. A joint trajectory allows fast searches of large areas. A cartesian trajectory will force the fork to travel in a straight line.

e-sonar# { **0 - 15** } - Sonar to use at execution time.
e-range { floating point } - Desired range of edge.
e-delta { floating point } - Desired magnitude of edge.
halt-enable { **true, false** } - The robot will come to a complete halt after this command if the flag is true.

3.3. Equate Path Point Command

equate | **e-1sonar#** | **e-2sonar#** |
| **e-threshold** | **spare** | **e-axis** | **halt-enable** |
| **loc phrase** |

| traj phrase |

The **equate** command is used to rotate the fork to achieve balanced range readings between the two sonars, **e-1sonar#** and **e-2sonar#**. The parameters command a rotation about the **e-axis** axis of the cartesian frame specified by the **loc phrase** (only the **tool nul** frame is supported) until the two selected sonars read the same, within **e-threshold**. Since the rotation commanded will be positive about an axis if **sonar#1**{range-value} - **sonar#2**{range-value} > 0 it is necessary to consider the order of the selected sonars. As an example if the programmer desired to use a pair of down-looking sonars to align with the floor, sonars 0 and 3 are used to command a rotation about the X axis (see Figure 3.1.). To achieve the proper direction parameter **sonar#1** is set to 3 and **sonar#2** is set to 0.

sonar#1	{ **0 - 15** }
sonar#2	{ **0 - 15** } - The two sonars to equate.
threshold	{ floating point } - Acceptable threshold for balanced sonars.
spare	{ **0.0** } - Not used.
axis	{ **X, Y, Z** } - The axis of rotation; one of the cartesian axes of the tool frame.
halt-enable	{ **true, false** } - The robot will come to a complete halt after this command if the flag is true.

3.4. Scan Path Point Command

scan **| s-select | skip-till | pallet-area |**
| closest-point-delta | spare | scan-return-pose-^ | halt-enable |
| loc phrase |
| traj phrase |

This is a high level Path Point command used to determine the nominal location of a rectangular object such as a 155mm pallet. The robot scans an area taking range readings starting from it's present position and ending at **loc phrase**. Each range reading is linked to the position of the robot when the sonars are read and then stored into the file **SONAR-FILE**. The **scan** command returns the nominal pose of the object in the record pointed to by **scan-return-pose-^** (Section 5.7.9.12. **RETURN-SCAN-POSE** describes the nominal pose of a pallet). If a pallet is not found the error **no-pallet** is returned.

s-select	{ **0 - 15** } - Sonar to use at execution time.
skip-till	{ integer } - Skip this number of valid sonar readings before storing the value with the robot pose. This conserves the number of readings taken but reduces the resolution of the scan.
pallet-area	{ floating point } - Used as the maximum range where the pallet can be from the sonar. The first and last readings of the scan less then **max-object-range** determine the right and left edge of the pallet respectively.
closest-point-delta	{ floating point } - Sets the delta of range values that can be grouped together with the closest range. This group then provides information such as whether a side or a corner of the pallet is the closest feature of the pallet to the robot (See 5.7.9.12. **RETURN-SCAN-POSE**).
spare	{ **0** } - Not used.
scan-return-pose-^	{ Ascii string } - Names a pose which **scan** sets to the nominal pose of the pallet as derived from the search. The parameter is the name of the pose, the compiler searches the **POSE-FILE** and stores the **record#** of the pose in the parameter. An error will be

returned if the pose has not been previously defined.

halt-enable { **true, false** } - The robot will come to a complete halt after this command if the flag is true.

3.5. Align-grip Path Point Command

align-grip | **perm-edge-val** | **perm-equal-val** |
spare	**side-equate-thresh**
max-short-side	**min-short-side**
tilt-ratio	**halt-enable**
loc phrase
goto traj phrase
edge traj phrase
equate traj phrase

This is a high level Path Point command which utilizes information from **scan** and current sonar data to determine if the long side of a 155mm pallet is facing the robot. It is assumed the fork is within 25 inches of the pallet and the orientation of the YZ plane of the fork with respect to the pallet falls into one of three cases: Long-case where the fork faces the pallet long side, Short-case where the fork faces the pallet short side and Tilt-case where the fork faces a corner of the pallet. **Edge**, **equate** and **goto** type commands are performed to roughly align the fork with the long side of the pallet.

perm-edge-val {floating point } - This value is added to the current sonar range value to set the range trigger for subsequent edge routine (See 5.7.10.7. **SHORT-SIDE-PROCESSING** and 5.7.10.8. **LONG-SIDE-PROCESSING**).

perm-equal-val {floating point } - Two sonar range values are considered equal if their difference is less than this value. Used in Long-case to verify that the fork is aligned with the pallet long side (See 5.7.10.8. **LONG-SIDE-PROCESSING**).

spare { **0.0** } - Not used.

side-equate-thresh { floating point } - Sets the threshold value for equate commands (see **equate**).

max-short-side { floating point } - Maximum length of pallet short side. Used in Long-case to test if short side of pallet is actually facing the fork (See 5.7.10.8. **LONG-SIDE-PROCESSING**).

min-short-side { floating point } - This parameter is the minimum measurement of the short side of a pallet. It is used if the pallet is not in Tilt-case to determine if the pallet is in Long or Short-case. If the sonar readings show that the measurement of the side perpendicular to the fork is greater then **min-short-side** then that side is the long side, ie the side parallel to the fork is the short side (See 5.7.10.4. **TEST-FOR-ALIGNMENT**).

tilt-ratio { integer value between **3** and **5** } - This is used to test if the pallet is tilted with respect to the fork. The Path Point command **scan** returns data regarding the left edge, the right edge and the closest feature of the pallet. Two values indicating the distance between the edges and the closest feature are compared and if their difference is greater then **tilt-ratio** then the probability is high that the pallet is tilted with respect to the fork (See 5.7.10.4. **TEST-FOR-ALIGNMENT**).

halt-enable { **true, false** } - The robot will come to a complete halt after this command if the flag is true.

3.6. Approach-pallet Path Point Command

approach-pallet | **t-x-sonar#** | **t-x-range** | **t-x-threshold** |
t-y-sonar#	**t-y-range**	**t-y-threshold**
r-x-sonar#1	**r-x-sonar#2**	**spare**	**r-x-threshold**
r-y-sonar#1	**r-y-sonar#2**	**spare**	**r-y-threshold**
loc phrase
traj phrase

This command handles the final approach to the 155mm pallet long side by achieving the proper height and orientation of the fork prior to inserting the tines beneath the pallet. Four tool motions are calculated in a manner similar to the **range** and **equate** Path Point commands. Ranges are specified by the **t-x-** and **t-y-** parameters signifying translation along the X and Y axes. Equates are specified by the **r-x-** and **r-y-** parameters signifying rotation about the X and Y axes. The parameters are the same as the parameters for **range** and **equate**.

t-x-sonar#	{ **0 - 15** } - Desired sonar for X axis ranging (see **range**).
t-x-range	{ floating point } - Desired sonar range to achieve.
t-x-threshold	{ floating point } - Acceptable threshold for desired range.
t-y-sonar#	{ **0 - 15** } - Desired sonar for Y axis ranging (see **range**).
t-y-range	{ floating point } - Desired sonar range to achieve.
t-y-threshold	{ floating point } - Acceptable threshold for desired range.
r-x-sonar#1	{ **0 - 15** }
r-x-sonar#2	{ **0 - 15** } - Desired sonars for X axis rotation (see **equate**).
spare	{ **0.0** } - Not used.
r-x-threshold	{ floating point } - Acceptable threshold for balanced sonars.
r-y-sonar#1	{ **0 - 15** }
r-y-sonar#2	{ **0 - 15** } - Desired sonars for Y axis rotation (see **equate**).
spare	{ **0.0** } - Not used.
r-y-threshold	{ floating point } - Acceptable threshold for balanced sonars.

3.7. Pickup-pallet Path Point Command

pickup-pallet | **p-p-sonar** | **p-p-sonar-offset** | **p-p-z-correction** |
| **spare** | **halt-enable** |
| **loc phrase** |
| **traj phrase** |

This command insures that the robot fork tines do not hit the 155mm pallet feet as the fork is inserted. Optical proximity sensors (see Figure 3.1) detect when the tines are too close to the pallet feet. The **p-p-sonar** sonar is used to initially obtain the distance the fork must travel before stopping the robot. The programmer can use the parameter **p-p-sonar-offset** to modify the distance the fork will travel to the pallet in the following manner: **p-p-sonar#**{value} - **p-p-sonar-offset** = distance to pallet. For example if **p-p-sonar-offset** equals 4.0, the fork will stop four inches from the pallet. When a proximity sensor senses a pallet foot the robot translates in the appropriate Z axis direction until the sensor is no longer active. Because of the sensor characteristics an additional translation can be specified in **p-p-z-correction** which forces the fork to travel further after the sensor has cleared the pallet foot.

p-p-sonar	{ **0 - 15** } - Desired sonar for determining distance to pallet load.
p-p-sonar-offset	{ floating point } - Desired sonar range to achieve.
p-p-z-correction	{ floating point } - Desired Z axis motion to be travelled after sensor clears the pallet foot. This parameter can be used to help center the fork

since there may be a large dead-band where none of the fork tine proximity sensors can detect the pallet feet.

spare { **0.0** } - Not used.

3.8. Goto-until-sw Path Point Command

goto-until-sw | **desired-sw** | **desired-sw-status** |
| **loc phrase** |
| **traj phrase** |

The **goto-until-sw** command operates similarly to the **goto** Path Point command except the motion is terminated if the selected switch specified by **desired-sw** achieves the desired condition as indicated by **desired-sw-status**. The error **point-reached** is returned if the robot reaches the goal specified by **loc phrase** prior to satisfying the switch condition.

switch { **0 - 7** } - The desired switch. Currently eight switches on the fork tines can be used. Additional switches can be integrated, up to sixteen total.

switch condition { **open, closed** } - The desired switch condition.

3.9.. Return-pose Path Point Command

return-pose | **return-pose-para-^** |

This command sets the pose record specified by **return-pose-para-^** (defined in the data dictionary) to the current pose of the robot. This command is used to dynamically store a pose after the robot has moved to a position (usually under sensor control).

return-pose-para-^ { ASCII string } - The name of the desired pose to initialize (must have been previously defined). The compiler searches the **POSE-FILE** and stores the **record#** of the pose in the parameter. An error will be returned at compile time if the pose has not been previously defined.

3.9.. Delay Path Point Command

delay | **delay-#-cycles** |

This command is used to introduce delays between the execution of other Path Point commands. An example of it's usage is to allow the fork to stabilize during complete halts. **delay-#-cycles** specifies the number of control cycles to delay before terminating the command.

delay-#-cycles { integer } - The number of control cycles to delay.

4. Programming the FMR Using RSL

The primary command for the FMR is **TRANSFER.** The **TRANSFER** command is decomposed by the TASK level into six steps which specify the paths for the pickup and release of an object. All six paths must be defined and compiled from the RSL level (RCS Manual Chapter 10 pages 10-8 thru 10-10) before the **TRANSFER** command is executed. The six paths are:

1) **move-to**
2) **approach-pickup**
3) **depart-pickup**
4) **move-to**
5) **approach-release**
6) **depart-release**

Two examples of RSL programs (consisting of the six paths) are now given. The first is a plan to locate and transfer a randomly oriented pallet from a truck bed to a conveyor. The second example is a plan to locate and transfer an array of pallets from a truck bed to a conveyor. A randomly oriented pallet has no constraints on which side of the pallet faces the robot and therefore the controller must determine the proper entry side. The array of pallets is constrained such that it's orientation does not vary from a predefined orientation by greater then 10 degrees and the entry side of each pallet within the array is known apriori.

4.1. Transfer of a Randomly Located 155mm Pallet

4.1.1. The RSL Plan for Random Pallet

The plan to transfer a 155mm pallet assumes a pallet is randomly positioned on a truck bed and the location of the truck and the conveyor have been taught before hand. Briefly the plan commands the robot controller to: align the fork with the truck bed, search for a 155mm pallet, determine the entry side of the pallet, insert the fork tines and finally place the pallet on the conveyor. Figure 4.1. shows a truck loaded with several pallets; the random pallet scenario handles only one pallet on the truck.

```
( This path moves the robot to a starting position before scanning the truck.)
-path-  move-to  PALLET 1  loc HOME  loc TRUCK

( Go to a predefined location where the truck bed will be using a fast joint
trajectory motion.)
-ppt-   goto     goal nul
                 joint 30.0 30.0 5.0

( Achieve proper roll orientation.)
-ppt-   equate   0 2 .5 .0 X true
                 tool nul
                 cart .2 .15 .25 .3 .2 .25

( Achieve proper pitch orientation.)
-ppt-   equate   3 0 .5 20.0 Z true
                 tool nul
                 cart .2 .15 .25 .3 .2 .25

( Achieve proper elevation from truck bed.)
-ppt-   range    0 18.00 .50 Y true
                 tool nul
```

```
                    cart .2 .15 .25 .3 .2 .25

( This path scans the truck from left to right searching for a single pallet,
aligns the fork with long side of pallet, and engages the fork.)
-path-  approach-pickup  PALLET 1  loc TRUCK

( Search for the pallet using a joint trajectory with SCAN-MTB defining the
destination of the search.)
-ppt-   scan        0  0  50.0  .5  0  PALLET-POSE true
                    tool SCAN-MTB
                    joint 25.0  25.0  5.0

( Scan has returned the nominal position of the pallet in PALLET-POSE.
Now go to the location PALLET-LOC which is defined as the pose
PALLET-POSE with an offset to safely position the fork in front of the
pallet.)
-ppt-   goto        loc PALLET-LOC
                    joint  25.0  25.0  1.0

( Make sure fork is within 25 inches of pallet before executing align-grip.)
-ppt-   range       1  20.00 1.0  X  true
                    tool nul
                    cart .2 .15 .25 .3 .2 .25

( Align with the pallet long side.)
-ppt-   align-grip
        4.0  8.0  0.0  .75  15.0  14.0  5
        true tool nul
        cart .3 .15 .25 .3 .15 .25
        cart .3 .15 .25 .3 .15 .25
        cart .3 .15 .25 .3 .15 .25

( Lower fork so that the lower base of the pallet acts as the sonar target.)
-ppt-   range       15  2.0  0.25  Y  true tool nul
                    cart .2 .15 .25 .3 .2 .25

( Achieve fine alignment with pallet entry side)
-ppt-   approach-pallet
        7   17.75 .5
        15  28.5   .25
        10  15 .0  .25
        7    6 .0  .5  true
        cart .3 .07 .25 .3 .07 .25

( Guide fork underneath the pallet until within range of sonar 12.)
-ppt-   pickup-pallet   7  4.0  0.5  true
                        cart .3 .3 .25 .3 .15 .25

( Use sonar to achieve precise servoed distance.)
-ppt-   range       12  3.0  .25  X  true
                    tool nul
                    cart .2 .15 .25 .3 .2 .25
```

```
( Move in the final distance to engage pallet.)
-ppt-    goto         tool PALLET-ENGAGE
                      cart .3 .3 .25 .3 .3 .25


( This path departs the from the truck by picking up the pallet and tilting it
back)
-path-   depart-pickup  PALLET 1 loc TRUCK

( Goto PALLET-UP1 which is an offset up from current position under
pallet.)
-ppt-    goto         tool PALLET-UP1
                      cart .3 .3 .25 .3 .3 .25

( PALLET-UP2 tilts pallet back.)
-ppt-    goto         tool PALLET-UP2
                      cart .3 .3 .25 .3 .3 .25


( The second move-to path describes a movement to an intermediate safe
position.)
-path-   move-to  PALLET 1 loc TRUCK  loc CONVEYOR

( CONV-SAVE is predefined safe position near the conveyor which can be
approached using a high joint velocity.)
-ppt-    goto         loc CONV-SAFE
                      joint 10.0 30.0 5.0


(  This path describes the approach to the conveyor and the release of the
pallet. CONVEYOR location has been taught beforehand.)
-path-   approach-release  PALLET 1  loc CONVEYOR

( CONV-UP1 is an offset up from CONVEYOR goal. Remember the goal
type takes the movetable, CONV-UP1, and adds it to the path goal,
CONVEYOR.)
-ppt-    goto         goal  CONV-UP1
                      joint  10.0  30.0  5.0

( CONV-UP2 removes the tilt of the pallet with respect to the CONVEYOR.)
-ppt-    goto         goal  CONV-UP1
                      cart .3 .3 .25 .3 .3 .25

( Place the pallet at the goal: CONVEYOR.)
-ppt-    goto         goal  nul
                      cart .3 .3 .25 .3 .3 .25

( Extract the fork tines.)
-path-   depart-release  PALLET 1  loc CONVEYOR

( CONV-BACK extracts the fork from the CONVEYOR.)
-ppt-    goto         goal  CONV-BACK
                      joint  10.0  30.0  5.0
```

4.1.2. The RSL Command to Transfer a Random Pallet

The command to transfer the pallet based on the plan described in Section 4.1.1. has the following syntax:

TRANSFER PALLET 1 loc TRUCK loc CONVEYOR

The command is entered from the TASK level (RCS Manual Chapter 9.4, page 9-7). The robot must currently be at the location **HOME** or a **nul-path** error will be generated (unless another path is mistakenly found).

4.2. Transfer of a Randomly Located Array of 155mm Pallets

4.2.1. RSL Plan to Transfer an Array of Pallets

The RSL plan to transfer an array of pallets illustrates some concepts of the NBS controller. The first concept is that of the array. The array is a data structure defined by the programmer from the RSL level (RCS Manual Chapter 10). For the FMR the array allows a collection of pallets to be organized as sectors in a structured manner.

The second concept is how the TASK level interprets the array structure during the execution of the **TRANSFER** command. The **TRANSFER** command syntax includes a field (the sector list) that allows the programmer to specify which pallets (sectors) and in what order the pallets (sectors) are to be transferred. If more than one pallet is to be transferred the Task level will step through the array. Each step through the array consists of using the array movetables, which give the dimensions of each sector, to re-position the fork between the transfer of each pallet.

Figure 4.1. depicts the scenario associated with the array transfer plan. A partially filled 2x3 array (sectors 1 and 5 are empty) is randomly situated on a truck bed. The FMR searches for sector 0 of the array. This sector is used to define the position and the orientation of the array. As stated earlier, all other sectors are related to the array defining sector by the array movetables. The plan takes into account minor deviations in position (plus/minus six inches) and orientation (plus/minus five degrees) in the placement of a pallet withing a sector. A more sophisticated plan would handle worse alignments but this results in increased sensor processing and consequently longer transfer cycle times.

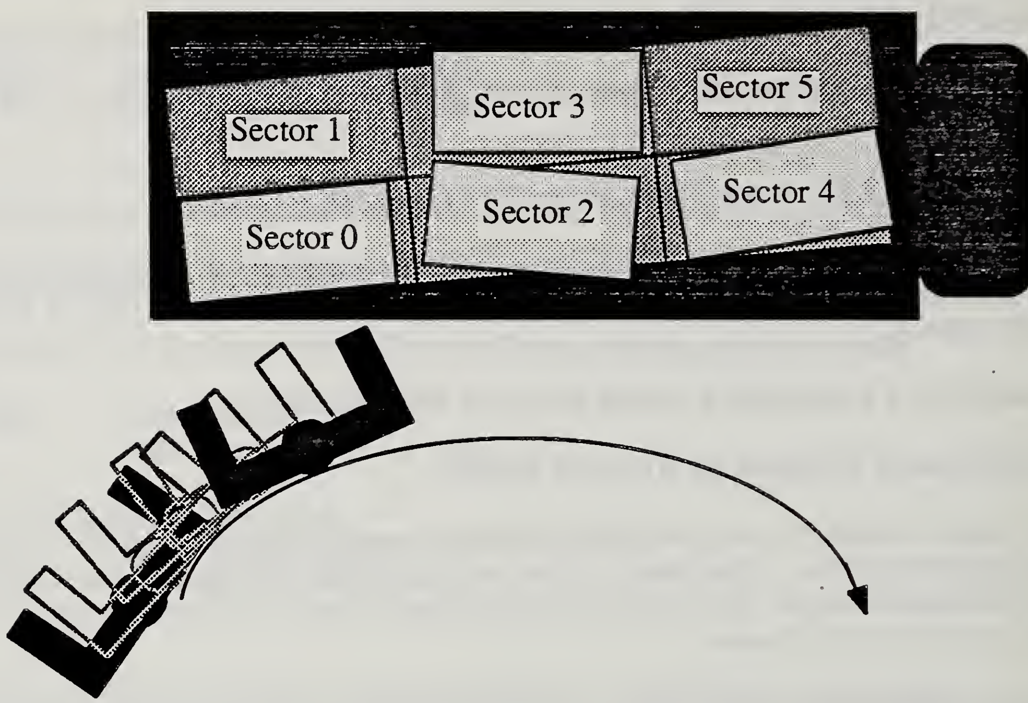


Figure 4.1. FMR searching for an array of 155mm pallets randomly situated on a truck.

Because the source of the **TRANSFER** command is an array (section 4.1.2. addresses the syntax of the **TRANSFER** command) all six paths are executed for each sector of the array to be transferred. There is one exception. Since the position of the array is not known beforehand the first iteration of the **TRANSFER** command must include a search for the array, or in other words, the search for sector 0 is different from the search for the remaining sectors. Two differing **move-to** paths are used to accommodate the initial array search from the subsequent pallet search operations. The **TRANSFER** command reconciles which path to choose using parameters from the command and the paths. In the example plan, the **move-to** which describes the path from the location **HOME** to the array **TRUCK** is executed the first time to search for the array. Each remaining sector uses the **move-to** which describes the path from the location **CONVEYOR** back to the array **TRUCK**. Note that this path takes advantage of the array structure to execute a minimum search in acquiring the pallet.

```
( Scan the truck from left to right searching for the left edge of the array,
align with the pallet in sector 1, engage the fork tines, and return the
location of the array)
-path-  move-to  PALLET 1  loc HOME  arr TRUCK

( Go to a predefined location where the truck bed will be using a fast joint
trajectory motion.)
-ppt-   goto     goal nul
                 joint 30.0 30.0 5.0

( Achieve proper roll orientation.)
-ppt-   equate   0 2 .5 .0 X true
                 tool nul
                 cart .2 .15 .25 .3 .2 .25
```

```
( Achieve proper pitch orientation.)
-ppt-   equate    3  0  .5  20.0  Z true
                  tool nul
                  cart  .2  .15  .25  .3  .2  .25

( Achieve proper elevation from truck bed.)
-ppt-   range     0  18.00 .50  Y  true
                  tool nul
                  cart  .2  .15  .25  .3  .2  .25

( Now look for the edge of the array.)
-ppt-   edge      1  50.0  .0  false
                  tool SCAN-MTB
                  joint 25.0  25.0  5.0

( Guarantee sonar 1 is on the first pallet.)
-ppt-   goto      loc tool -CONE
                  joint  25.0  25.0  1.0

( Since two sonars, 1 and 5, are on pallet align with pallet.)
-ppt-   equate    1  5  .5  0.0  Y true
                  tool nul
                  cart  .2  .15  .25  .3  .2  .25

( Make sure fork is within 25 inches of pallet.)
-ppt-   range     1  20.00 1.0  X  true
                  tool nul
                  cart  .2  .15  .25  .3  .2  .25

( Bring sonar 8 onto the pallet.)
-ppt-   edge      8  28.0  .0  false
                  tool -TOOL
                  cart  .2  .15  .25  .3  .2  .25

( Lower fork so that the lower base of the pallet acts as the sonar target.)
-ppt-   range     15  2.0  0.25  Y  true
                  tool nul
                  cart  .2  .15  .25  .3  .2  .25

( Achieve fine alignment with pallet entry side)
-ppt-   approach-pallet
        7   17.75  .5
        15  28.5   .25
        10  15 .0  .25
        7   6  .0  .5   true
        cart  .3  .07  .25  .3  .07  .25

( Guide fork underneath the pallet until within range of sonar 12.)
-ppt-   pickup-pallet   7  4.0  0.5  true
                        cart  .3  .3  .25  .3  .15  .25

( Use sonar to achieve precise servoed distance.)
-ppt-   range     12  3.0  .25  X  true
```

```
                      tool nul
                      cart .2 .15 .25 .3 .2 .25

( Move in the final distance to engage pallet.)
-ppt-    goto         tool PALLET-ENGAGE
                      cart .3 .3 .25 .3 .3 .25

( Now return pose which defines the location of the array.)
-ppt-    return-pose      TRUCK


( This path is used to engage the fork with all remaining pallets in each
sector of the array TRUCK. It is executed after the first pallet is transferred
to the conveyor.)
-path-  move-to PALLET 1 loc CONVEYOR arr TRUCK

( Move to a position in front of next sector of array. Note that the TASK
level will automatically index to the next sector because 1, the path
destination is an array and 2, the path point location type is goal.)
-ppt-    goto         goal SECTOR-SAFE
                      joint 25.0 25.0 5.0

( Lower fork so that the lower base of the pallet acts as the sonar target.)
-ppt-    range        15 2.0 0.25 Y true
                      tool nul
                      cart .2 .15 .25 .3 .2 .25

( Achieve fine alignment with pallet entry side)
-ppt-    approach-pallet
         7   17.75 .5
         15 28.5   .25
         10 15 .0 .25
         7   6 .0 .5 true
         cart .3 .07 .25 .3 .07 .25

( Guide fork underneath the pallet until within range of sonar 12.)
-ppt-    pickup-pallet  7 4.0 0.5 true
                        cart .3 .3 .25 .3 .15 .25

( Use sonar to achieve precise servoed distance.)
-ppt-    range        12 3.0 .25 X true
                      tool nul
                      cart .2 .15 .25 .3 .2 .25

( Move in the final distance to engage pallet.)
-ppt-    goto         tool PALLET-ENGAGE
                      cart .3 .3 .25 .3 .3 .25


( The approach to the pallet is handled by the initial move-to, thus this path
is empty.)
-path-  approach-pickup  PALLET 1  arr TRUCK

( All paths require at least one path point command, in this case a goto tool
```

```
nul.)
-ppt-   goto     tool nul
                 cart .3 .3 .25 .3 .3 .25
```

The **depart-pickup, move-to, approach-release** and **depart-release** are identical to the corresponding paths for the randomly oriented pallet in Section 4.1.1.

4.2.2. The RSL Command to Transfer an Array of Pallets

The command to transfer an array of pallets based on the plan described in Section 4.2.1. has the following syntax:

TRANSFER PALLET 1 arr TRUCK 0 2 4 3 ; loc CONVEYOR

The command is entered from the TASK level (RCS Manual Chapter 9.4, page 9-7). The robot must currently be at the location **HOME** or a **nul-path** error will be generated (unless another path is mistakenly found). The sector list for the array **TRUCK** in the above command shows how individual pallets are selected for transfer and in what order. The commanded transfer sequence for the array is sectors 0, 2, 4 and 3.

5. The FMR PATH Level

This section serves as an augmentation to The NBS Real-time Control System User's Reference Manual section on the PATH level (section 10.5) and closely follows the format of that section.

5.1. PATH Commands

FMR conforms to this section.

5.2. PATH Input Command Buffer

FMR conforms to this section.

5.3. PATH Status Information

The **status-arg-out** values have been extended for the FMR. The status values are now centrally located in the **VAR-O error-list** (located in block 10 of the RSL level). This can be viewed in show mode from any level. The complete list of the status values and their meaning follow.

System errors

0 **noerror**

-1 **programmer** - Programming error. Check last piece of code compiled.

-2 **command-error** - invalid command.

TASK errors

101 **task-cmd** - invalid argument in command.

102 **nul-path** - path can't be found. Hasn't been loaded or defined by the user.

103 **path-error** - error reported by PATH level.

PATH errors

200 **path-cmd** - invalid argument in command.

201 **ppt-para** - invalid Path Point command or parameter.

202 **prim-error** - error reported by PRIM level.

203 **sensor-cond** - robot paused because some sensor condition was not met. The PATH level variable **ppt-command** will show the current Path Point command which reported the error.

204 **point-reached** - reached goal before sensor condition met. Usually means an edge condition was not met before the goal was reached. Look at the **VAR-O edge-var** and **edge-para** to determine which edge conditions were not met.

205 **no-echo-sonar** - sonar reading shows no object detected.

206 **sonar-limit** - object is closer then lower limit of the selected sonar.

207 **no-pallet** - the **scan** routine did not detect a pallet. This is normally caused by the pallet being outside the **pallet-area** parameter (set when the **scan** path point is compiled; try raising the limit) or by the pallet right edge not being detected (move pallet closer to the beginning of the scan not towards the goal destination of the scan or increase the rotation of the scan destination movetable).

208 **sw-bad** - indicates a faulty switch. In theory the switches cannot detect both pallet feet simultaneously. Thus, if switches on both tines are active, one switch must be faulty. There may be dirt on the lens, or worse the switches are actually seeing both pallet feet because of a misalignment of the switches, the sensitivity of a switch has increased or the pallet feet are not the expected distance apart.

209 **no-sonar** - a sonar has been selected but the model for the sonar has not been compiled. Check to see if the entry exists for the sonar by showing **snr-model-array** on the

PATH level. A **0** in an element indicates there is no entry for the particular sonar.

5.4. PATH Errors

FMR conforms to this section.

5.5. PATH Processing

FMR conforms to this section.

5.6. PATH Preprocessing

The routine that reads the sonars has been added to the preprocess portion of the PATH control cycle. The sonars have several modes of operation which are described in Section 5.6.1.2. The programmer selects the desired sonar to read and waits for the range data to arrive. This process is described in section 5.6.1.

5.6.1. SONAR-READ

Input: **sonar-request**
Output: **sonar-range, sonar-valid, echo-valid, low-lim**

A routine that wishes to read a particular sonar(s) need only set the element of the **sonar-request** array corresponding to the desired sonar to **true**. The smacro statement to select sonar 0 is:

true => sonar-request { 0# }

would enable the sonar. The variable **sonar-valid** will return the value **true** when the sonar's range data has been set. The data is stored in the array **sonar-range** and can be accessed from within a routine in the following manner (again for sonar 0);

sonar-range { 0# } .=>. temporary-storage

or from the terminal in show mode as such.

:S sonar-range

SONAR-READ tests **status**, which if equal to **error**, will not over-write the sonar values during the error cycle. Physically, the desired sonars are enabled by **SELECT-SONAR** which also sets the bits in the variable **curr-sel-sonar** to one for each sonar selected. If a sonar is selected, the routine **READ-SONAR-DATA** is called.

The following routines are associated with the sonar sensors.

5.6.1.1. SELECT-SONAR

Input: **prev-sel-sonar, prev-sel-mode**
Output: **curr-sel-sonar, curr-sel-mode**

The primary responsibility of this routine is to physically control the sonar sensors. **GET-SONAR-SELECTION** is called to set **curr-sel-sonar**. If **curr-sel-sonar** is not equal to **prev-sel-sonar** then the user has changed the selection of the sonar. This logic is also tested against **curr-sel-mode** which determines in what mode the user desires the sensors to be run (see 5.6.1.2. **RESET-SONAR**). **RESET-SONAR** is called if the same sonar needs to be fired again (by having its status cleared) or if the user's sensor requirements have changed.

5.6.1.2. RESET-SONAR

Input: **flood-sonar**
Output: sonar selection and mode

This routine calls the routines that write to the sensor electronics and initializes key variables. If the sonars are operating in flood fashion, that is all sonars fire continuously but the ones selected via **sonar-request** are the only ones read, then there is no need to call **OUTPUT-SONAR-SELECTION.** In this case, the sonars are turned on once by the routine **FLOOD-SONAR.**

The two modes the sonars operate in are free-run (sonars are fired continuously via external clock) and one-shot (sonars are fired once via user software). The modes can run in either flood fashion where all sonars are physically fired, or non-flood fashion where only the sonars selected via **sonar-request** are physically fired. The sonars are typically used in a flood fashion which increases the probability that a signal from one of the sonars (not necessarily a **sonar-request** sonar) will reflect off a surface and be processed by a selected sonar. Note that the reading may not be exact but it will allow the robot to proceed and possibly extract more accurate information.

The mode is set using **OUTPUT-SONAR-MODE.** The variable **sonar-on,** when written to port **j1-c**, resets all the sonar status bits. This is needed before the next echo can be received.

5.6.1.3. OUTPUT-SONAR-SELECTION

Input: **curr-sel-sonar**
Output: sonar selection

Writes the low byte of **curr-sel-sonar** out to port **j1-a** and then the high byte.

5.6.1.4. OUTPUT-SONAR-MODE

Input: **curr-sel-mode**
Output: mode

Writes the variables which contains the code for the desired sonar mode to port **j1-c.**

5.6.1.5. READ-SONAR-DATA

Input: **sonar-rdy-status, #of-cycle-waiting**
Output: **sonar-valid, need-one-shot**

This routine keeps track of when to read the sonars. The status of each sonar currently selected is set from **GET-SONAR-STATUS. sonar-rdy-status** has each bit set to one for the appropriate sonar having valid range data. This is compared to **curr-sel-sonar** (since they both keep track of the desired sonar in a bit map fashion) to find out when all the sonars are ready to be read. Some sonars may not return a ready status ever (the echo is never received). To prevent an endless loop situation, **#of-cycle-waiting** keeps track of how many cycles have passed. If it exceeds **#of-cycle-allowed** then the waiting stops and the sonars that have valid readings as well as the errant sonars are all set using **SET-SONAR-RANGE.** Finally **sonar-valid** is set true to indicate that the sonar values in **sonar-range** are now valid, and **need-one-shot** is set true which allows **SELECT-SONAR** to clear the status bit for the next range reading.

5.6.1.6. GET-SONAR-STATUS

Input: **curr-sel-sonar**
Output: **sonar-rdy-status**

The status of sonars 0 thru 7 are read in and then sonars 8 thru 15. They are masked with **curr-sel-sonar** to remove status information from sonars that have not been selected by the user. Status will arrive from unwanted sonars when they are operated in a flood fashion since all sonars are fired in this case. The correct status is returned in **sonar-rdy-status**.

5.6.1.7. SET-SONAR-RANGE

Input: **curr-sel-sonar, sonar-rdy-status, snr-model-array**
Output: **sonar-range**

This routine sets the range values of all sonars selected by the user regardless if an echo has arrived or not. Each sonar is tested to see if it has been selected. If so the sonar model is retrieved. If a status has arrived for the sonar then the raw range data is retrieved using **GET-SONAR-DATA**. The raw value is then converted and stored into **sonar-range**. If the converted range is less then the minimum detectable range then the flag **min-limit** is set **true**. If the status bit indicates no echo from the sonar then the maximum range of the sonar, **snr-max-range**, is stored in **sonar-range**.

5.6.1.8. GET-SONAR-DATA

Input: **sonar-index**
Output: **J2-A**

This routine reads one sonar range value from the sonar interface box. **sonar-index** selects which sonar to read. **jj** is set according to **sonar-index**, a 0 if **sonar-index** is between 0 and 7 or a 1 if between 8 and 15. **J1-B** is set to select the sonar within the two groups (0-7 or 8-15). If the low group is chosen, the high byte is selected and read in, then the low byte is selected, read in and added to the high byte. If the high group is chosen, the high byte is selected and read in, then the low byte is selected, read in and added to the high byte. The raw range data is returned in **J2-A**.

5.6.1.9. CLEAR-SONAR-REQUEST

This routine sets all elements of the **sonar-request** array to false, clearing any requests.

5.6.1.10. SONAR-OFF

This routine clears all sonar requests and shuts off all the sonars.

5.7. PATH Decision Processing

The bulk of the FMR application resides in the routines that execute the path points. These routines which are an extension to the routines listed in the RCS Manual are covered in this section.

5.7.1. SEND-HALT

Input: **halt-enable, halt-request, prim-status**
Output: **ppt-done**

This routine is used by other Path Point commands to halt the robot and is highly tailored for this purpose. A **HALT** will be executed if the **halt-enable** flag equals **true** (this is done when the path point parameter is compiled). The **halt-request** variable is primarily used to ensure that only one **HALT** is executed. **HALT** selects the desired halt pose. PRIM will report done when it has reached the halt pose. Since this routine is used quite often by other path point routines the flag **ppt-done** is set **true** (if the halt pose is reached while the sensor condition is still met) at the termination of this routine.

5.7.2. HALT

Input: **tool-pose-^, traj-type**
Output: **prim-com-pose, traj-para**

This routine commands PRIM to halt at the current position as defined by the pose stored in **prim-com-pose**. The halt pose is dependent on the commanded trajectory since PRIM will place the tool pose at the commanded PRIM pose for a cartesian trajectory and will place the sensor at the commanded PRIM pose for a servo trajectory.

The current pose of the tool frame is first retrieved. If the trajectory type is servo then the sensor movetable is added to the tool pose and stored in **prim-com-pose**, else the tool pose is stored in **prim-com-pose.**

5.7.3. TRANSLATE

Input: **tool-pose-^, delta, axis**
Output: **prim-com-pose**

This routine constructs a pose in **prim-com-pose** that is a translation of **delta** along the tool frame **axis**.

5.7.4. ROTATE

Input: **tool-pose-^, delta, axis**
Output: **prim-com-pose**

This routine constructs a pose in **prim-com-pose** which is a rotation of **delta** about the pallet frame **axis**. An attempt has been made to modify the rotation such that the positional relationship

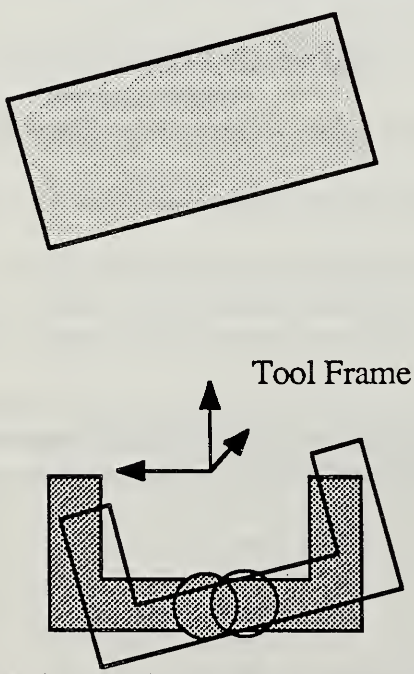
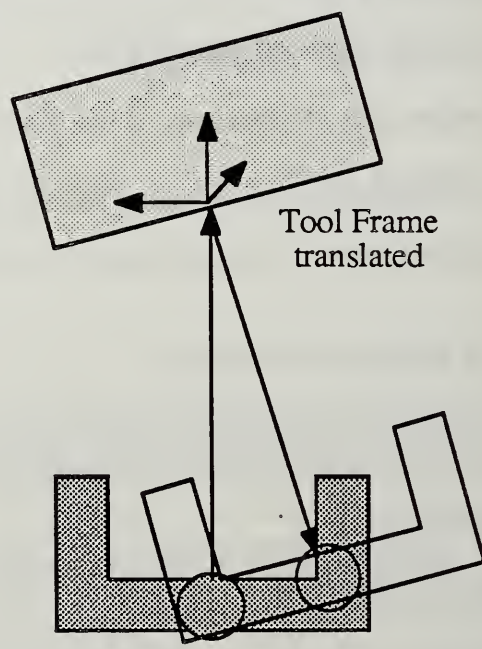


Figure 5.1a. ROTATE about tool frame



5.1b. ROTATE about translated tool fame

between the pallet and the fork does not change. Figure 5.1a shows a rotation motion about the tool frame. Figure 5.1b shows a more desirable motion taking place about the pallet frame by preceding the rotation by a translation (derived from sensor range data). This maintains the proper relationship between the fork mounted sensors and the pallet.

5.7.5. INIT-SONAR-MODEL-ARRAY
Input: **SONAR-MODEL-FILE**
Output: **snr-model-array**

This routine retrieves the record numbers of the sonar model entries in the file **SONAR-MODEL-FILE** and stores them in the array **snr-model-array**. The sonar model variable **snr-id** is used to index into the **sonar-model-array** such that the pointer to the sonar model record for sonar number 1 is found in element 1 of **snr-model-array**. For information on the sonar model itself see section 2.1. Sonar Modelling.

5.7.6. Range Path Point Routines

The following routines are associated with the RANGE Path Point command (section 3.1.).

5.7.6.1. RANGE-PPT
Input: **new-ppt**
Output: RANGE routines

This is the high level routine called during the execution of the RANGE Path Point command. The first time executed it initializes all variables, all remaining executions call **RANGE**.

5.7.6.2. RANGE-PPT-INIT
Input: **RANGE-FILE**
Output: **range-para, TRAJ-PHRASE, halt-enable, halt-request, sonar-request**

This routine retrieves the parameters compiled in the Path. It uses these parameters to turn on the needed sonars and to retrieve the trajectory parameters.

5.7.6.3. RANGE-SONAR-MODEL-INIT
Input: **snr-model-array**
Output: **snr-model-para**

This routine retrieves the sonar model for the chosen range sonar.

5.7.6.4. RANGE
Input: **sonar-valid, min-limit, range-para**
Output: **delta, axis, halt-request, TRANSLATE, RANGE-GOAL, PRIM-TRAJ, SEND-HALT**

This routine generates an error value, **delta**, as the difference (in inches) between the sonar range reading and the desired range reading (**r-range**).. If **min-limit** equals **true** then this routine will not function properly and therefore reports an error. This can happen, for example, if a Polaroid sensor is chosen instead of a Migatron sensor to range to a value of 2 inches. If the sensor is at 5 inches, a forward motion is commanded. But since the value does not change (because the Polaroid sensor will not operate at 5 inches) this routine continues to command forward motion.

If **delta** is less then **r-thresh** then **SEND-HALT** is called to bring the robot to a halt. If **delta** exceeds **r-thresh** then depending on the trajectory type **RANGE-GOAL** or **TRANSLATE** is called to calculate a new goal pose which will minimize the difference between the desired range and the actual range. Because the robot can overshoot the goal which will cause the sonar readings to again generate an error value greater then **r-thresh**, this routine may be called several times. In this case **halt-request** is set **false** which clears any previous calls to **SEND-HALT**.

5.7.6.5. RANGE-GOAL
Input: **tool-pose-^, snr-mtb-^, axis, delta**
Output: **prim-com-pose**

This routine constructs a pose in **prim-com-pose** that is a translation of **delta** along the sonar frame **axis.** It is used to position the sonar at the goal pose as opposed to **TRANSLATE** which positions the tool frame at the goal pose.

5.7.7. EDGE Path Point Routines

The following routines are associated with the EDGE Path Point command (section 3.2.).

5.7.7.1. EDGE-PPT
Input: **new-ppt**
Output: EDGE routines

This is the high level routine called during the execution of the EDGE Path Point command. The first time executed it initializes all variables, all remaining executions call **EDGE.**

5.7.7.2. EDGE-PPT-INIT
Input: **EDGE-FILE**
Output: **edge-para, TRAJ-PHRASE, LOC-PHRASE, halt-enable, halt-request, sonar-request**

This routine retrieves the parameters compiled in the Path. It uses these parameters to turn on the needed sonars and to retrieve the trajectory and location parameters.

5.7.7.3. EDGE-SONAR-MODEL-INIT
Input: **snr-model-array**
Output: **snr-model-para**

This routine retrieves the sonar model for the chosen edge sonar.

5.7.7.4. EDGE
Input: **sonar-valid, range-set, range-para**
Output: **current-range, initial-range, ?EDGE, SEND-HALT**

This routine tests for the conditions that satisfy a perceived edge. The first valid sonar reading is saved in **initial-range.** This is required to satisfy the first edge condition that a change in range between the first reading and the current reading must exceed **e-delta.** Once the value is saved **PRIM-TRAJ** is called to start the robot moving to the goal pose (a parameter of EDGE). The second edge condition requires that the range values, starting with **initial-range** and ending at the **current-range,** must cross the value given in the parameter **e-range.** This allows the user to specify at what range to look for the edge. Figure 5.2 shows an example of an edge observed at a range of 50 inches. This edge was distinguished from the other edges by selecting an **e-delta** of 30 inches and an **e-range** of 60 inches. Both conditions are tested and if they are true **SEND-HALT** is called to bring the robot to a halt. If they are not true then **?EDGE** is called to determine if an error has occurred.

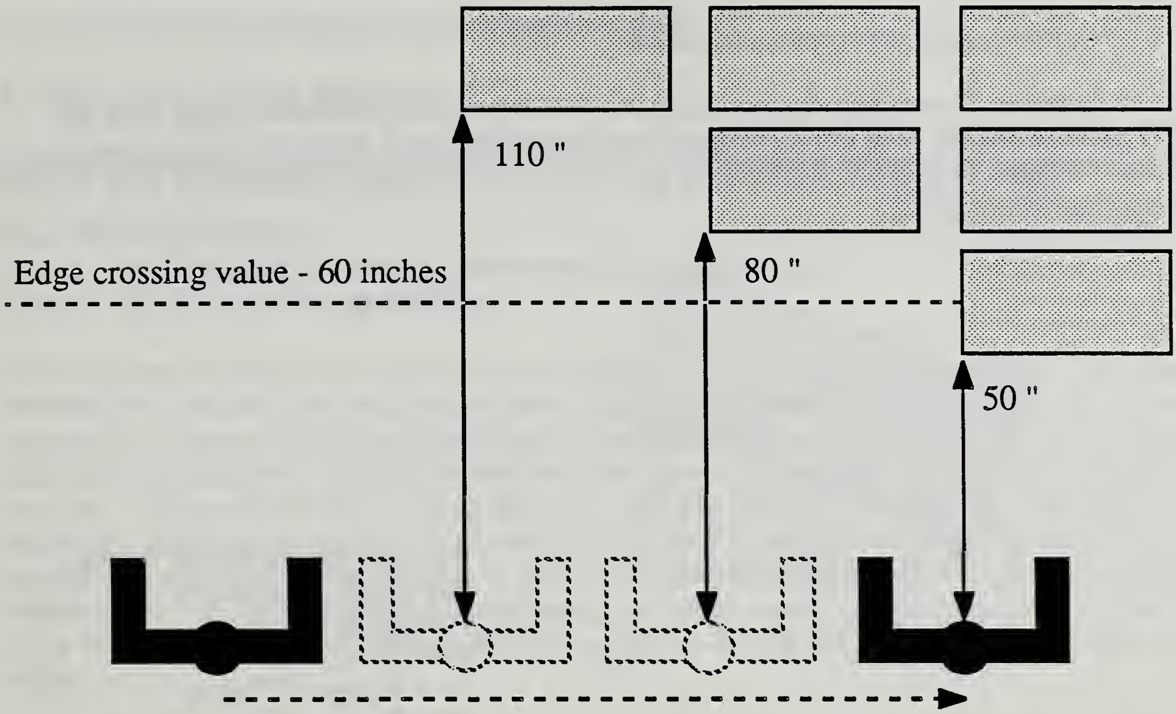


Figure 5.2. An edge threshold of 60 inches "filters" the edge from 110 inches to 80 inches but passes the edge from 80 inches to 50 inches during the sensor scan.

5.7.7.5. ?EDGE
Input: **prim-status, halt-request**
Output: **PRIM-PAUSE, status, status-arg**

This routine tests the status from the PRIM level to determine if the robot has reached the goal pose without the edge conditions being satisfied. If this occurs the error **sensor-cond** is returned in **status-arg**.

5.7.8. EQUATE Path Point Routines

The following routines are associated with the EQUATE Path Point command (section 3.3.).

5.7.8.1. EQUATE-PPT
Input: **new-ppt**
Output: EQUATE routines

This is the high level routine called during the execution of the EQUATE Path Point command. The first time executed it initializes all variables, all remaining executions call **EQUATE**.

5.7.8.2. EQUATE-PPT-INIT
Input: **EQUATE-FILE**
Output: **equate-para, axis, TRAJ-PHRASE, halt-enable, halt-request, sonar-request**

This routine retrieves the parameters compiled in the Path. It uses these parameters to turn on the needed sonars, set the axis of rotation and retrieve the trajectory parameters.

5.7.8.3. EQUATE-SONAR-MODEL-INIT
Input: **snr-model-array, snr-model-para, e-axis**

Output: **snr-model-para, snr-sprtn, snr-offst**

This routine retrieves the models for both sonars. The data from the model along with the EQUATE parameters are used to determine the geometry required for rotation using the range data from two sonars. Figure 5.3 shows the geometry for two sonar rotation (the axis of rotation is perpendicular to the page).

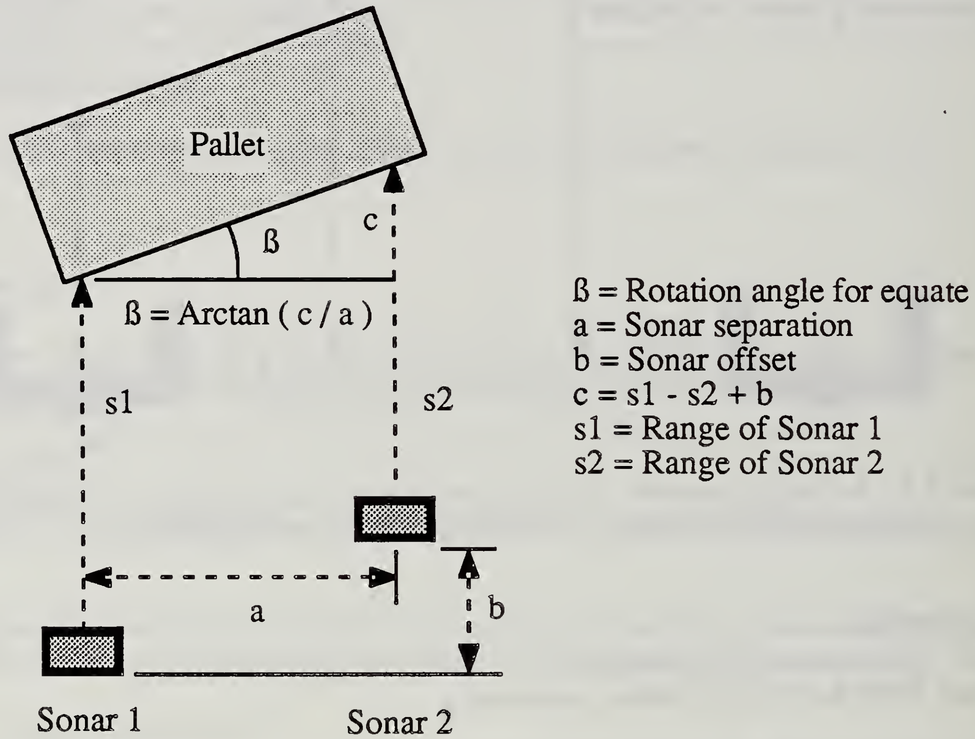


Figure 5.3. Geometry for EQUATE rotation.

After the sonar models have been retrieved, the position of each sonar is extracted and stored in the vectors **el-xyz** and **e2-xyz**. The vectors are to extract the rotation parameters: sonar separation (**snr-sprtn**) and sonar offset (**snr-offst**).

5.7.8.4. EQUATE

Input: **sonar-valid, equate-para**
Output: **SEND-HALT, NOT-EQUATED**

This routine generates an error value, **eq-delta**, as the difference (in inches) between the sonar readings. If **eq-delta** is less then **e-threshold** then **SEND-HALT** is called to bring the robot to a halt. If **eq-delta** exceeds **e-threshold** then **NOT-EQUATED** is called to calculate a new goal pose which will minimize the difference between the sonars.

5.7.8.5. NOT-EQUATED

Input: **eq-delta, snr-sprtn**
Output: **delta, ROTATE, PRIM-TRAJ**

This routine calculates the angle **delta** which is the arctangent of the quotient **eq-delta** and **snr-sprtn**. The **delta** is clipped to within plus or minus **eq-max**. Because the robot can overshoot the goal which will cause the sonar readings to again generate an error value greater then **e-threshold**, this routine may be called several times. Thus, **halt-request** is set **false**

which clears any previous calls to **SEND-HALT**.

5.7.9. SCAN Path Point Routines

The following routines are associated with the SCAN Path Point Command (section 3.4.).

5.7.9.1. SCAN-SONAR

Input: **new-ppt, prim-status, more-records, right-edge-^**
Output: SCAN routines, **ppt-done**

This routine commands the robot to search the work volume for a pallet load. The search path is selected by the user but the direction must be from left to right (although this can be easily changed). Sonar readings and positions of the robot during the read are stored for analysis. The first time the routine is called all necessary variables are initialized and the robot is started in motion. The scan terminates when the robot reaches the goal or when the right edge of the pallet is found. If the right edge is not found and the goal is reached a **no-pallet** error is returned in **status-arg** (pallet may be to close to the start or the end of the search path). The pointer **right-edge-^** is set to the record containing the range value and position of the robot when the edge was observed. It remains a **0#** (nul) until the **SAVE-SCAN-ROUTINE** detects the right edge.

Once the right edge has been detected **RETURN-SCAN-POSE** is called to write out the pose indicating the closest feature of the pallet to the robot. **N1-N2-SET** is called to return values needed by the ALIGN-GRIP Path Point command.

5.7.9.2. SCAN-INIT

Input: **SCAN-FILE**
Output: **scan-para, LOC-PHRASE, TRAJ-PHRASE, halt-enable, sonar-request, pallet-min-range, SONAR-FILE-INIT, PRIM-TRAJ**

This routine retrieves the parameters compiled in the Path. It uses these parameters to turn on the selected sonar and to retrieve the trajectory and location parameters. **SCAN-VAR-INIT** is called to initialize pertinent variables. **SONAR-FILE-INIT** initializes the file that stores the scan readings.

5.7.9.3. SCAN-SONAR-MODEL-INIT

Input: **snr-model-array**
Output: **snr-model-para**

This routine retrieves the sonar model for the chosen scan sonar.

5.7.9.4. SAVE-SCAN-READING

Input: **sonar-valid, skip-reading, skip-till**
Output: **skip-reading, ADD-TO-SONAR-REC**

This routines saves the scan reading by calling **ADD-TO-SONAR-REC** to store the sonar reading and actual robot pose to file **SONAR-REC**. Variable **skip-till** allows valid sonar readings to be skipped to conserve memory.

5.7.9.5. ADD-TO-SONAR-REC

Input: **left-edge-^, right-edge-^, ass/rec, max/rec**
Output: **ADD-RECORD, SET-PALLET-MIN-RANGE, LEFT-EDGE-TEST, RIGHT-EDGE-TEST**

This routine is responsible for storing the scan data and determining an estimate of the position of the pallet. **ADD-RECORD** stores the sonar range value and the pose of the robot into the file **SONAR-FILE**. **SET-PALLET-MIN-RANGE** ensures that the minimum range reading during the scan is stored in **pallet-min-range** and **temp-min-pose-^** points to the record that contains the scan reading. **LEFT-EDGE-TEST** tests for the left edge of the pallet and returns the number of the scan record associated with the edge in **left-edge-^**.
RIGHT-EDGE-TEST performs the same function for the pallet right edge, the scan record is returned in **right-edge-^**.

5.7.9.6. ADD-RECORD

Input: **sonar-range, tool-pose-^, s-ptr**
Output: **save-sonar-var, save-range**

This routine stores the sonar range and the tool pose into the **save-sonar-var** record of file **SONAR-FILE**. The pointer **s-ptr** keeps track of the next available record in the file. This prevents the routine **add-record** from searching the file for a new record. Each record includes a field called **back-link** which is used to link the file backwards. This assists in file manipulations in later routines.

5.7.9.7. SET-PALLET-MIN-RANGE

Input: **save-range, pallet-min-range, s-ptr**
Output: **pallet-min-range, temp-min-pose**

This routine ensures that **pallet-min-range** contains the minimum range reading during the scan and that **temp-min-pose-^** points to the associated **SONAR-FILE** record. The initial value of **pallet-min-range** is initialized to the parameter **pallet-area** in **SCAN-INIT**.

5.7.9.8. LEFT-EDGE-TEST

Input: **save-range, pallet-area, left-edge-^, l-e-cnt, l-e-thresh, s-ptr**
Output: **l-e-cnt, left-edge-^, LEFT-EDGE-ADJUST**

This routine tests for the left edge of the pallet. A sonar range less then **pallet-area** indicates the possibility of the pallet edge. To filter out any noise typically generated from the sonar side lobes several valid reading are required before the edge is accepted. A count of the edge readings is kept in **l-e-cnt**. If the count exceeds the threshold in **l-e-thresh** then **left-edge-^** is set to the current record associated with the edge reading. Since the reading is actually the last in a sequence of valid edge readings **LEFT-EDGE-ADJUST** is called to reset **left-edge-^** to the first valid reading. **l-e-cnt** is reset to **0#** whenever a value greater then **pallet-area** is encountered to ensure that the sequence of edge values are all valid (ie. one false reading starts the edge filtering algorithm over).

5.7.9.9. LEFT-EDGE-ADJUST

Input: **left-edge-^, l-e-cnt, pallet-area**
Output: **left-edge-^, d0l**

This routine resets **left-edge-^** to the first valid reading of the pallet left edge. **Left-edge-^** initially contains the last reading of the edge and is used to start the backwards search in **SONAR-FILE**. The search terminates when a range reading greater then **pallet-area** is encountered. The record which has the first valid edge reading is stored into **left-edge-^**. The range value of the edge is stored into **d0l**. The coordinates of the edge can be obtained by transforming the pose in the field **scan-pose** a distance of **d0l** in the plus X direction.

5.7.9.10. RIGHT-EDGE-TEST

Input: **save-range, pallet-area, left-edge-^, right-edge-^, r-e-cnt, r-e-thresh, s-ptr**
Output: **r-e-cnt, left-edge-^, RIGHT-EDGE-ADJUST**

This routine tests for the right edge of the pallet. It operates in a manner similar to **LEFT-EDGE-TEST** except it test for range values greater then **pallet-area.**

5.7.9.11. RIGHT-EDGE-ADJUST

Input: **right-edge-^, r-e-cnt, pallet-area**
Output: **right-edge-^, d0r**

This routine resets **right-edge-^** to the last valid reading of the pallet left edge. It operates in a manner similar to **LEFT-EDGE-ADJUST** except the terminating condition of the edge search is a range value less then **pallet-area.**

5.7.9.12. RETURN-SCAN-POSE

Input: **temp-min-pose-^**
Output: **FIND-LEFT-CORNER, FIND-RIGHT-CORNER, SET-RETURN-POSE**

This routine is responsible for refining the position of the pallet and returning the pallet pose. The pallet pose is derived from the scan information. The scan routine was not intended to be the final routine used for determining the entry side of the pallet. It has not been investigated to see if this determination can be made in all cases. Therefore the pallet pose is the pose of the scan where the closest feature of the pallet was observed plus a few adjustments.

One adjustment is made to accommodate the three types of possible orientations that can exist between the pallet and the robot; tilt, long side and short side. The closest feature of the pallet when it is tilted (a corner) can be accurately derived but for the long and short side cases it is desirable to attain the center of the pallet side. Since the pose defining the side may not be the center an algorithm is used to derive the center. Figure 5.4a-c shows three examples of pallet orientation with all the features labeled. The short side case looks similar to the long side case shown in Figure 5.4b-c and therefore is not shown. The pose pointed to by **min-pose-^** is the actual pallet pose returned.

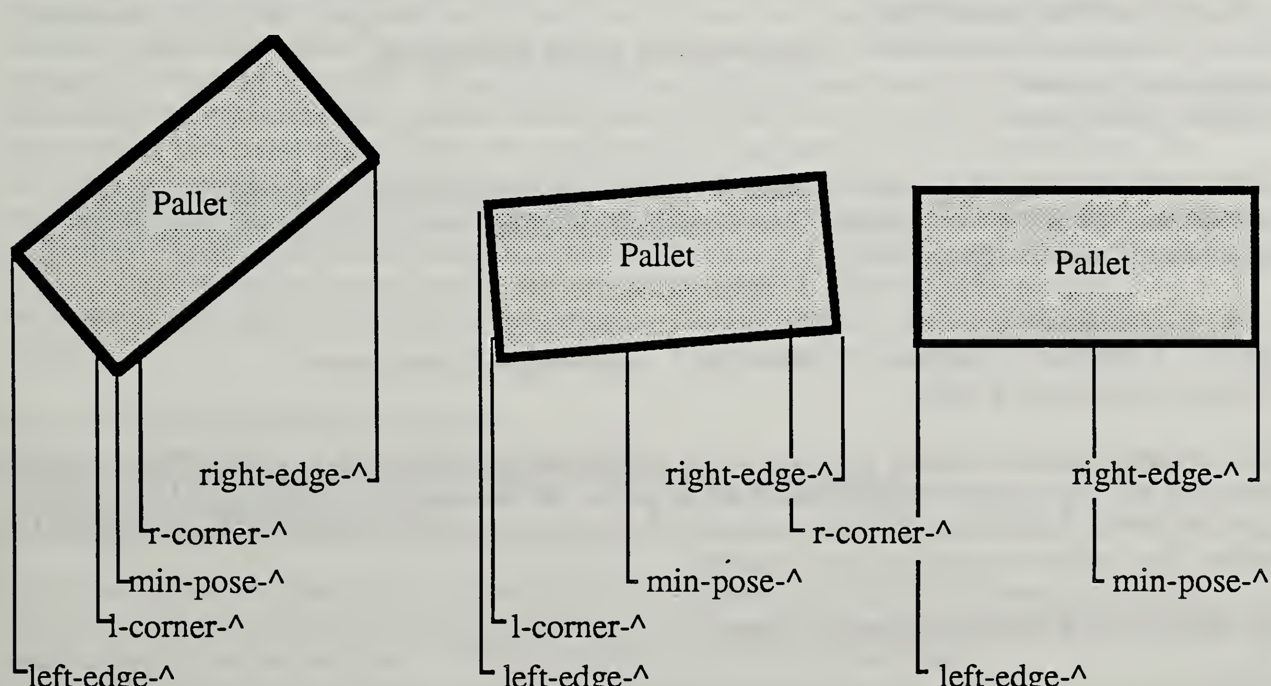


Figure 5.4a. Pallet Tilt Case. 4b. Pallet Long Case with slight tilt. 4c. Pallet Long Case with no tilt.

The first step of the center detection algorithm is to group together a set of scan values that are within a user selectable threshold (**closest-point-delta**) of the closest range observed. The end points of the group are stored in **r-corner-^** and **l-corner-^** (Figure 5.4a-c). The routines **FIND-LEFT-CORNER-EDGE** and **FIND-RIGHT-CORNER-EDGE** calculate these values.

The only remaining step in the algorithm is to find the middle entry in the group of scan values. The routine **SET-RETURN-POSE** locates the middle entry (**min-pose-^**) and retrieves the scan pose. The scan pose is translated out along the X axis of the tool frame a distance specified by the parameter **pallet-min-range** and then written out to the pose record pointed to by the parameter **scan-return-pose-^**.

5.7.9.13. FIND-LEFT-CORNER-EDGE

Input: **temp-min-pose-^, pallet-min-range, save-sonar-var, scan-para**
Output: **l-corner-^**

This routine searches SONAR-FILE from **temp-min-pose-^** (points to record containing **pallet-min-range**) backwards finding the last pallet reading that is within the parameter **closest-point-delta** of **pallet-min-range**. This left edge of the closest point is pointed to by **l-corner-^**.

5.7.9.14. FIND-RIGHT-CORNER-EDGE

Input: **temp-min-pose-^, pallet-min-range, save-sonar-var, scan-para**
Output: **r-corner-^**

This routine searches forward from **temp-min-pose-^** using the same scheme as **FIND-LEFT-CORNER-EDGE** to set **r-corner-^** to the right edge of closest point.

5.7.9.15. SET-RETURN-POSE

Input: **r-corner-^, l-corner-^, bytes/record, pallet-min-range, scan-return-pose-^**
Output: **min-pose-^**

This routine bisects the group of records to isolate the center of the closest pallet feature. **min-pose-^** points to this record. The pose is also written out to the pose pointed to by the parameter **scan-return-pose-^**.

5.7.9.16. N1-N2-SET

Input: **r-corner-^, l-corner-^, left-edge-^, right-edge-^, min-pose-^**
Output: **n1, n2, n3, n1+n2**

This routine performs further analysis of the pallet scan. **n1** equals the number of readings taken from the left edge to the closest feature of the pallet, **n2** the number of readings from the right edge to closest feature and **n3** is the number of readings associated with the closest feature of the pallet. The values are used in later routines.

5.7.10. ALIGN-GRIP Path Point Routines

The following routines are associated with the ALIGN-GRIP Path Point command (section 3.5.).

5.7.10.1. ALIGN-GRIP
Input: **new-ppt, l-done-flag, test-called, sonar-valid**
Output: **GRIP-RETRIEVE, TURN-ON-SONAR, TEST-FOR-ALIGNMENT, FIND-GRIP-SIDE-STATE, ppt-done**

This is the high level routine called during the execution of the ALIGN-GRIP Path Point command. The first time executed it initializes all variables and turns on the required sonars. The terminating condition for this routine is when **l-done-flag** equals **true** which means the fork is aligned with the long side of the pallet. **TEST-FOR-ALIGNMENT** is called one time to estimate the initial orientation of the pallet. The remaining executions wait for valid sonar data and then call **FIND-GRIP-STATE** to determine if a robot motion is required to align the fork with the pallet.

5.7.10.2. GRIP-RETRIEVE
Input: **ALIGN-GRIP-FILE**
Output: **align-grip-para, halt-enable, halt-request**

This routine retrieves the parameters compiled in the Path.

5.7.10.3. TURN-ON-SONAR
Input:
Output: **sonar-request**

This routine turns on sonars 1, 5 and 8.

5.7.10.4. TEST-FOR-ALIGNMENT
Input: **n1, n2, d0l, d0r, pallet-min-range**
Output: **align-grip-para, s-done-flag, t-done-flag, l-done-flag, l-flag, s-flag, t-flag, step-state, first-time, test-called**

This routine is called one time to calculate the nominal orientation of the pallet. The pallet orientation which describes what the robot saw during the most recent scan falls into three categories; short side, long side and tilted. In the short and long side cases the closest pallet feature (**pallet-min-pose**) is the center of the side facing the robot; in the tilt case it is the corner of the pallet facing the robot. **n1** and **n2** (the number of sonar readings between the edges and the closest point) is used to separate the tilt case from the short/long side cases. If the difference between **n1** and **n2** is greater then the parameter **tilt-ratio** then the tilt case exists. To distinguish between the short and long side cases the distance between the closest pallet feature (**pallet-min-range**) is subtracted from the maximum range reading between **d0r** and **d0l** (distance to edges) which effectively calculates the side of the pallet perpendicular to the robot. This is then compared to the parameter **min-short-side**. If the side is larger than **min-short-side** then the short side case exists, else the long side of the pallet is facing the robot.

5.7.10.5. FIND-GRIP-SIDE-STATE
Input: **s-done-flag, t-done-flag, l-done-flag**
Output: **SHORT-SIDE-PROCESSING, TILT-PROCESSING, LONG-SIDE-PROCESSING**

This routine uses **l-done-flag**, **s-done-flag** and **t-done-flag** to call the three processing routines. The processing routines are set up as a sequence of steps which read sonars, make decisions and move the robot. **SHORT-SIDE-PROCESSING** and **TILT-SIDE-PROCESSING** move the robot to the long side of the pallet.

LONG-SIDE-PROCESSING then aligns the robot with the pallet.

5.7.10.6. TILT-PROCESSING

Input: **n1, n2, step-state, ppt-done**
Output: **SIDE-MOVE, SIDE-EQUATE, sonar-request, ppt-done, t-done-flag, t-flag**

This routine assumes that sonar 1 (the center of the fork) is facing the closest corner of the pallet and the angle of the pallet is such that the larger of **n1** and **n2** represents the long side. **0-step** tests **n1** and **n2** and sets the **+zdirection** true if the long side is in the positive Z direction of the tool frame and false otherwise. The parameters are set for the proceeding steps. **1-step** moves to clear the corner so that **SIDE-EQUATE** in step 2 will be facing a side. **2-step** then equates with the long side. Now the robot is in a position to execute **LONG-SIDE-PROCESSING.**

5.7.10.7. SHORT-SIDE-PROCESSING

Input: **d0l, d0r, step-state, sonar-range, align-grip-para**
Output: **SIDE-MOVE, SIDE-EQUATE, sonar-request, ppt-done, s-done-flag, s-flag**

This routine assumes that sonar 1 is centered on the short side of the pallet. The greater range reading between **d0l** and **d0r** is assumed to be directly related to the long side of the pallet. **0-step** tests this and set the parameters and **+zdirection. 1-step** takes sonar 1 to the edge (corner) adjacent to the chosen long side. When done it sets **movtab-^** to the necessary movetable needed to rotate the fork around so that it is roughly parallel to the long side. A larger rotation movetable is used if the process flow has come from **LONG-SIDE-PROCESSING** because it has mistaken the short side as a long side. Experience has shown this occurs most often when the side is nearly parallel to the fork and thus the need for a larger rotation. A **SIDE-EQUATE** had been used initially to rotate about the corner but it's execution was not reliable. **2-step** executes the rotation move. **3-step** then aligns the fork with the long side. Now the robot is in a position to execute **LONG-SIDE-PROCESSING.**

5.7.10.8. LONG-SIDE-PROCESSING

Input: **d0l, d0r, step-state, sonar-range, align-grip-para**
Output: **SIDE-EDGE, SIDE-EQUATE, sonar-request, ppt-done, l-done-flag, l-flag**

This routine accomplishes the final alignment with the long side of the pallet. It has been extended to handle the error case of a short side being mistaken for a long side. **step-0** starts by determining where the processing flow has come from. If it came directly from **TEST-FOR-ALIGNMENT** as a long side case it tests the sonars against the parameter **perm-equal-val** to determine if any alignment is necessary. If one of the sonars is off the pallet then further alignment is needed. Also since it is possible that a short side case could exist a move is set up to align the outer sonars with what should be the long side edge of the pallet. If after aligning one sonar on the pallet edge the other sonar is off the pallet then the short side case actually exists. If processing came from tilt or short case then it is not necessary to test for this error (**1-step** is not called).

1-step forces the previously selected sonar off the pallet. When done it sets the parameters needed to bring the sonar back on to the pallet.

2-step uses **SIDE-EDGE** to bring the sonar on to the pallet. Further alignment is not required if the tilt or short case existed. If long side exists then the possibility of a short case error needs to be tested by turning on the necessary sonars (which takes an additional cycle) and calling **3-step**.

3-step tests the differences between sonars 1 and 2 against parameter **max-short-side**. If the difference is greater (a sonar is off the pallet) then a short side pallet case exists and **SHORT-SIDE-PROCESSING** must be executed, else the fork is aligned with the long side and processing is done.

5.7.10.9. SIDE-EQUATE
Input: **first-time**
Output: EQUATE routines

This routine is similar in execution to the EQUATE Path Point command. The first time executed it initializes all variables, all remaining executions call **EQUATE.**

5.7.10.10. SIDE-EQUATE-INIT
Input: **align-grip-para**
Output: **e-threshold, e-axis, axis, TRAJ-PHRASE**

This routine initializes the parameters needed for the **EQUATE** routine.

5.7.10.11. SIDE-EDGE
Input: **first-time**
Output: EDGE routines

This routine is similar in execution to the EDGE Path Point command. The first time executed it initializes all variables, all remaining executions call **EDGE.**

5.7.10.11. SIDE-EDGE-INIT
Input: **align-grip-para, align-mtb-^**
Output: **e-delta, range-set, halt-request, LOC-PHRASE, TRAJ-PHRASE**

This routine initializes the parameters needed for the **EDGE** routine.

5.7.10.12. SIDE-MOVE
Input: **first-time**
Output: **SIDE-MOVE-INIT, ppt-done**

This routine is similar in execution to the GOTO Path Point command. The first time executed it initializes all variables. It then waits for PRIM to report done.

5.7.10.13. SIDE-MOVE-INIT
Input: **align-grip-para, align-mtb-^**
Output: **e-delta, range-set, halt-request, LOC-PHRASE, TRAJ-PHRASE**

This routine initializes the parameters and calls **PRIM-TRAJ.**

5.7.10.14. -Z-DIRECTION
Input:
Output: **e-1sonar#, e-2sonar#, e-sonar, sonar-request, align-mtb-^, +zdirection**

This routine sets up the **SIDE-EDGE** move for motion in the minus Z direction in step 1 of **SHORT-SIDE-PROCESSING.**

5.7.10.15. +Z-DIRECTION
Input:

Output: **e-1sonar#, e-2sonar#, e-sonar, sonar-request, align-mtb-^, +zdirection**

This routine sets up the **SIDE-EDGE** move for motion in the plus Z direction in step 1 of **SHORT-SIDE-PROCESSING.**

5.7.11. APPROACH-PALLET Path Point Routines

The following routines are associated with the APPROACH-PALLET Path Point command (section 3.6.).

5.7.11.1. APPROACH-PALLET

Input: **new-ppt, sonar-valid, y-r-done, x-r-done, y-t-done, x-t-done**
Output: APPROACH-PALLET routines, **SEND-HALT, PRIM-TRAJ**

This is the high level routine called during the execution of the APPROACH-PALLET Path Point command. The first time executed it initializes all variables. The remaining executions wait for valid sonar data to determine if a robot motion is required to correct the fork position.

Four degrees of freedom are tested to determine if the fork is in the proper position and orientation with respect to the pallet. **CALC-Y-ROT** and **CALC-X-ROT** determine if there are deviations in orientation about the Y and X axes respectively (see Figure 3.1). With the addition of the Migatron sensors a test for orientation about the Z axis can also be incorporated. Care must be taken though that the sensors will always have a target to view (ie. a sensor isn't looking off the edge of the truck bed for instance). Position deviation along the Y and X axes are handled by **CALC-Y-TRANS** and **CALC-X-TRANS**. If the status from these routines report done then **SEND-HALT** is called. If any of the routines report not done then **PRIM-TRAJ** is called to move the robot.

5.7.11.2. APPROACH-PALLET-INIT

Input: **APPROACH-PALLET-FILE**
Output: **approach-pallet-para, TRAJ-PHRASE, halt-enable, halt-request, sonar-request**

This routine retrieves the parameters compiled in the Path. It uses these parameters to turn on the needed sonars and to retrieve the trajectory parameters.

5.7.11.3. APPROACH-PALLET-S-M-INIT

Input: **snr-model-array**
Output: **snr-model-para, x-r-sprtn, x-r-offst, y-r-sprtn, y-r-offst, y-t-offst, x-t-offst**

This routine retrieves the sonar model for the chosen sonars. Note that the routines **EQUATE-SONAR-MODEL-INIT** and **RANGE-SONAR-MODEL-INIT** are used since **APPROACH-PALLET** basically duplicates the **EQUATE** and **RANGE** commands.

5.7.11.4. CALC-Y-ROT

Input: **approach-pallet-para, sonar-range, y-r-offst, y-r-sprtn, prim-com-pose**
Output: **y-r-done, prim-com-pose**

This routine sets **y-r-done** true if the difference between the two selected sonars is less then the parameter **y-r-thresh**. If the difference is greater, the correction rotation is calculated as the arctangent of the quotient **delta** and **y-r-sprtn**. The correction is applied as a rotation to **prim-com-pose**. Note that the routine terminates after it is successful once. This will sacrifice accuracy for speed.

5.7.11.5. CALC-X-ROT

Input: **approach-pallet-para, sonar-range, x-r-offst, x-r-sprtn, prim-com-pose**
Output: **x-r-done, prim-com-pose**

This routine sets **x-r-done** true if the difference between the two selected sonars is less then the parameter **x-r-thresh**. If the difference is greater, the correction rotation is calculated as the arctangent of the quotient **delta** and **x-r-sprtn**. The correction is applied as a rotation to **prim-com-pose**. Note that the routine terminates after it is successful once. This will sacrifice accuracy for speed.

5.7.11.6. CALC-Y-TRANS

Input: **approach-pallet-para, sonar-range, y-t-offst, prim-com-pose**
Output: **y-t-done, prim-com-pose**

This routine sets **y-t-done** true if the difference between the actual sonar range and the desired sonar range is less then the parameter **y-t-thresh**. If the difference is greater, the correction translation is calculated as the difference between the two ranges. The correction is applied as a translation to **prim-com-pose**. Note that the routine terminates after it is successful once. This will sacrifice accuracy for speed.

5.7.11.7. CALC-X-TRANS

Input: **approach-pallet-para, sonar-range, x-t-offst, prim-com-pose**
Output: **x-t-done, prim-com-pose**

This routine sets **x-t-done** true if the difference between the actual sonar range and the desired sonar range is less then the parameter **x-t-thresh**. If the difference is greater, the correction translation is calculated as the difference between the two ranges. The correction is applied as a translation to **prim-com-pose**. Note that the routine terminates after it is successful once. This will sacrifice accuracy for speed.

5.7.12. PICKUP-PALLET Path Point Routines

The following routines are associated with the PICKUP-PALLET Path Point command (section 3.7.).

5.7.12.1. PICKUP-PALLET

Input: **new-ppt, range-set, switches, z-correcting, step-state**
Output: PICKUP-PALLET routines, **step-state**

This is the high level routine called during the execution of the PICKUP-PALLET Path Point command. The first time executed it initializes all variables and retrieves the sonar model. All remaining executions guide the robot fork beneath the pallet using a sonar to initially determine the distance to travel and the proximity sensors to determine if any corrective translations are required. The algorithm to insert the tines requires that a goal pose is calculated first. If a switch indicates an obstruction (closed) then the fork is moved sideways until the switch is clear (open). After the switch is clear an additional sideways motion can be user specified to accommodate variations in the sensitivity and orientation of the proximity sensors. Before the fork is moved sideways though the distance the fork has travelled since the last starting pose, and the remaining distance to travel is calculated. The remaining distance to travel is then used to generate the new goal pose after the sideways correction has been made. The algorithm terminates when a forward motion is completed (no more travel distance to the pallet) without a switch detecting an obstruction.

The routine **WAIT-RANGE-SET** sets **range-set** to **true** when the sonar has returned a range reading indicating the distance the sensor must travel to complete the command. Once the range has been determined **SWITCH-READ** is called to return the condition of the proximity switches.

Two tool frame motions are executed during this routine; forward translation in the X direction, and sideways translation in the Z direction. If a switch is closed (indicating proximity to an obstacle) then **UPDATE-DISTANCE-TO-GOAL** is called (only once according to **step-state**) to determine the distance the fork has travelled and how much distance remains. **FORK-ALIGNMENT** then handles the sideways motion until the switch opens indicating the obstacle has been cleared. With no obstacles the fork can now be moved forward towards the new goal by calling **MOVE-TO-PALLET**, but first the user specified offset in the Z direction is executed by **Z-CORRECTION** (again the process flow is handled by **step-state**).

5.7.12.2. PICKUP- PALLET-INIT

Input: **PICKUP-PALLET-FILE**
Output: **pickup-pallet-para, axis, TRAJ-PHRASE, halt-enable, halt-request, sonar-request, first-time, range-set, z-correcting, step-state, distance-traveled, distance-to-goal**

This routine retrieves the parameters compiled in the Path. It uses these parameters to turn on the needed sonars and to retrieve the trajectory parameters.

5.7.12.3. PICKUP-PALLET-SONAR-MODEL-INIT

Input: **snr-model-array**
Output: **snr-model-para**

This routine retrieves the sonar model for the chosen pickup-pallet sonar.

5.7.12.4. WAIT-RANGE-SET

Input: **pickup-pallet-para, sonar-valid, min-limit, sonar-range**
Output: **distance-to-goal, range-set**

This routine initializes the value of **distance-to-goal** using a parameter selected sonar sensor. The **sonar-valid** flag indicates when valid range data is available. If the flag **min-limit** equals **true** then the selected sonar will not work (either because it points in the wrong direction or it is operating outside it's intended range). The parameter **p-p-sonar-offset** is added to the sonar range so that the fork can be offset from the range calculated goal. **range-set** is set to true when the distance has been determined.

5.7.12.5. MOVE-TO-PALLET

Input: **first-time, tool-pose-^, distance-to-goal, prim-status**
Output: **TRANSLATE, PRIM-TRAJ, SEND-HALT**

This routine commands the robot to move the fork towards the pallet. The goal is calculated as the current tool frame translated a distance equal to **distance-to-goal** in the plus X direction. **PRIM-TRAJ** is called to command the robot to move the fork. If **prim-status** reports **done** then the goal has been reached and **SEND-HALT** is called to stop the robot. Note that the process flow will not reach this routine whenever a switch detects an obstacle during the move to the goal.

5.7.12.6. Z-CORRECTION

Input: **first-time, delta, p-p-z-correction**
Output: **TRANSLATE, PRIM-TRAJ, SEND-HALT, first-time, z-correcting**

This routine commands the robot to move the fork in a sideways manner after the proximity switches have cleared an obstacle. The goal is calculated using the parameter **p-p-z-correction** as the offset to be traveled. The direction is determined from the sign of **delta**. **TRANSLATE** is called to generate the goal pose and **PRIM-TRAJ** is called to command the robot to move the fork. **prim-status** reports **done** when the goal has been reached.

5.7.12.7. UPDATE-DISTANCE-TO-GOAL

Input: **tool-pose-^, t-pose, distance-to-goal**
Output: **distance-traveled, distance-to-goal**

This routine calculates the remaining distance to the goal. The current tool pose is compared to a saved tool pose stored in **t-pose**. The distance between the poses is calculated and stored in **distance-traveled**. That value is subtracted from the old **distance-to-goal** to formulate the new **distance-to-goal**.

5.7.12.8. FORK-ALIGNMENT

Input: **switches, l-tine-mask, r-tine-mask, l-tine-offset**
Output: **delta, SWITCH-CORRECTION**

This routine is called when a proximity switch detects an obstacle. The switch input is masked to decouple the left and right tine proximity data. Because of the geometry of the switches and the pallet, both tines cannot simultaneously detect a pallet. If this situation arises the **sw-bad** error is reported in **status-arg** (check if there is dirt covering the detector or examine the appropriate LED in the Proximity Interface Box to determine if the sensor is working).

Two tine calibration tables (**l-tine-calib-table** and **r-tine-calib-table**) are used to model the 155mm pallet foot (see section 2.2. Proximity Detector Modelling). The tables are calibrated to deliver the proper motion to clear the pallet feet dependent on which switches are active. The tables are organized such that the switch data can be used as the index into the table. There are several impossible switch active combinations that are flagged in the tables using a **0#** in the elements. As an example if the current left tine combination equaled 1001, (the outside switches are active), element 17 (1001) would have a **0#** indicating this is impossible. But if the tine combination equaled 0011 (the two right side switches are active) then element 3 (0011) would have a value indicating a translation direction and distance (-0.75 for example) needed to clear the switches. Thus the routine is divided to test the individual tine switches. The appropriate table signals an error or provides the motion value. An additional offset is provided to allow correction of all the calibration table values. **r-tine-offset** and **l-tine-offset** serve this function. They are initialized in the power-up block. **SWITCH-CORRECTION** is called to command the robot to move the fork and thereby clear the obstacle.

5.7.12.9. SWITCH-CORRECTION

Input: **switches, prev-switches, delta**
Output: **TRANSLATE, PRIM-TRAJ, prev-delta, z-correcting, first-time**

This routine commands the robot to move the fork to clear a proximity detected obstacle. The variables **switches** and **prev-switches** ensure that the same move isn't commanded every cycle, only with the occurrence of a new combination of switches. **delta** provides the translation direction and distance. Since this routine is only called when the switches are active, if the goal is reached (when **prim-status** reports **done**) an error of **point-reached** is returned. Typically this means that the table value given for clearing the obstacle (dependent on the combination stored in **switches**) is incorrect. The value or the offset value (**r-tine-offset** and **l-tine-offset**) can be made larger but care must be taken not to drive the fork too far such that the opposite tine proximity sensors detect the opposite pallet foot. If this happens the robot can

become unstable, oscillating between both pallet feet.

5.7.12.9. SWITCH-READ

Input: **switches**
Output: **switches, prev-switches**

This routine reads in the physical state of the proximity switches. The value of the old switch readings is saved in the variable **prev-switches**. The low byte and the high byte is read in and stored into **switches**. The value is complemented since the port uses an inverter.

5.7.13. The following routines are associated with the GOTO-UNTIL-SW Path Point command (section 3.8.).

5.7.13.1. GOTO-UNTIL-SW

Input: **new-ppt, switches, desired-sw-status**
Output: EDGE routines

This is the high level routine called during the execution of the GOTO-UNTIL-SW Path Point command. The first time executed it initializes all variables. During all remaining executions it monitors the **desired-sw** until it matches the **desired-sw-status**. If the PRIM level reports that the robot has reached the goal pose and the desired switch condition hasn't occurred then an error of **no-pallet** is reported in **status-arg**. If the desired switch condition occurs then **SEND-HALT** is called.

5.7.13.2. GOTO-UNTIL-SW-INIT

Input: **GOTO-UNTIL-SW-FILE**
Output: **sw-para, TRAJ-PHRASE, LOC-PHRASE, PRIM-TRAJ, halt-enable**

This routine retrieves the parameters compiled in the Path. It uses these parameters to retrieve the trajectory and location parameters. Finally **PRIM-TRAJ** is called to command the robot to move to the goal pose.

5.7.14. RETURN-POSE Path Point Routines

The following routine is associated with the RETURN-POSE Path Point command (section 3.9.).

5.7.14.1. RETURN-POSE

Input: **RETURN-POSE-FILE**
Output: **ppt-done**

This is the high level routine called during the execution of the RETURN-POSE Path Point command. It retrieves the parameter compiled in the Path, **return-pose-para-^**, which is a pointer to a pose record. The current tool frame is then written into the pose record.

5.7.15. DELAY Path Point Routines

The following routines are associated with the DELAY Path Point command (section 3.10.).

5.7.15.1. DELAY

Input: **delay-#-cycles**
Output: **delay-ctr, ppt-done**

This is the high level routine called during the execution of the DELAY Path Point command. The first time executed it initializes all variables. During the remaining executions it increments

delay-ctr until the variable equals **delay-#-cycles**. This introduces a delay between the execution of path point commands.

5.7.15.2. DELAY-INIT
Input: **DELAY-FILE**
Output: **delay-para, delay-ctr**

This routine retrieves the parameter compiled in the Path.

5.8. PATH Postprocessing

FMR conforms to this section.

5.9. Display and Debug Routines

This section does not appear in the RCS User's Manual. The purpose of the section is to present routines that are helpful in diagnosing problems with the NBS Sensor Package.

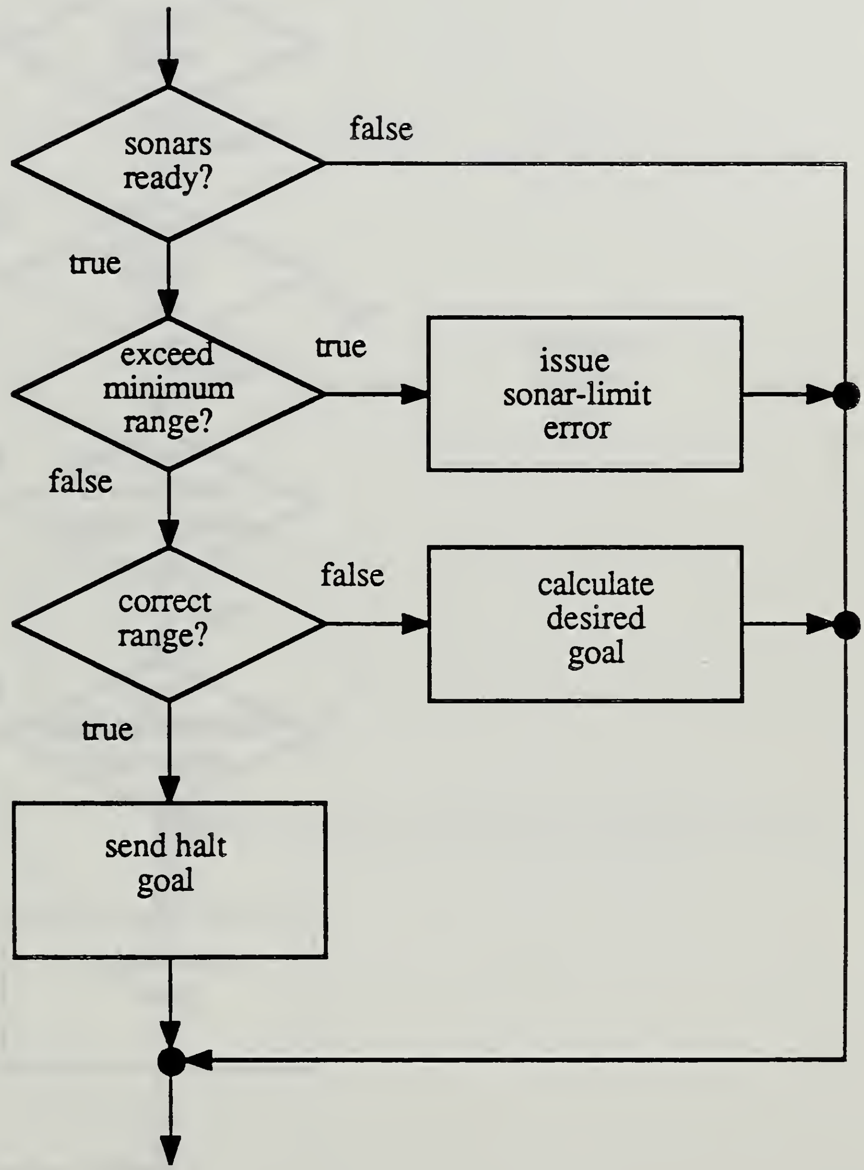
5.9.1. DISPLAY-SONAR-SELECTION
Input: **sonar-valid**
Output: **sonar-range**

This routine displays all sonars range values that have been requested (ie..the **sonar-request** array element for a sonar has been set to **true**). It should not be compiled into the background control loop for two reasons: First, **~PRINT** does not work in a background task and second, the routine polls on **sonar-valid** and will stall the control loop until the sonar echo arrives. This routine is primarily used for user verification of working sonars and can be executed directly from the terminal.

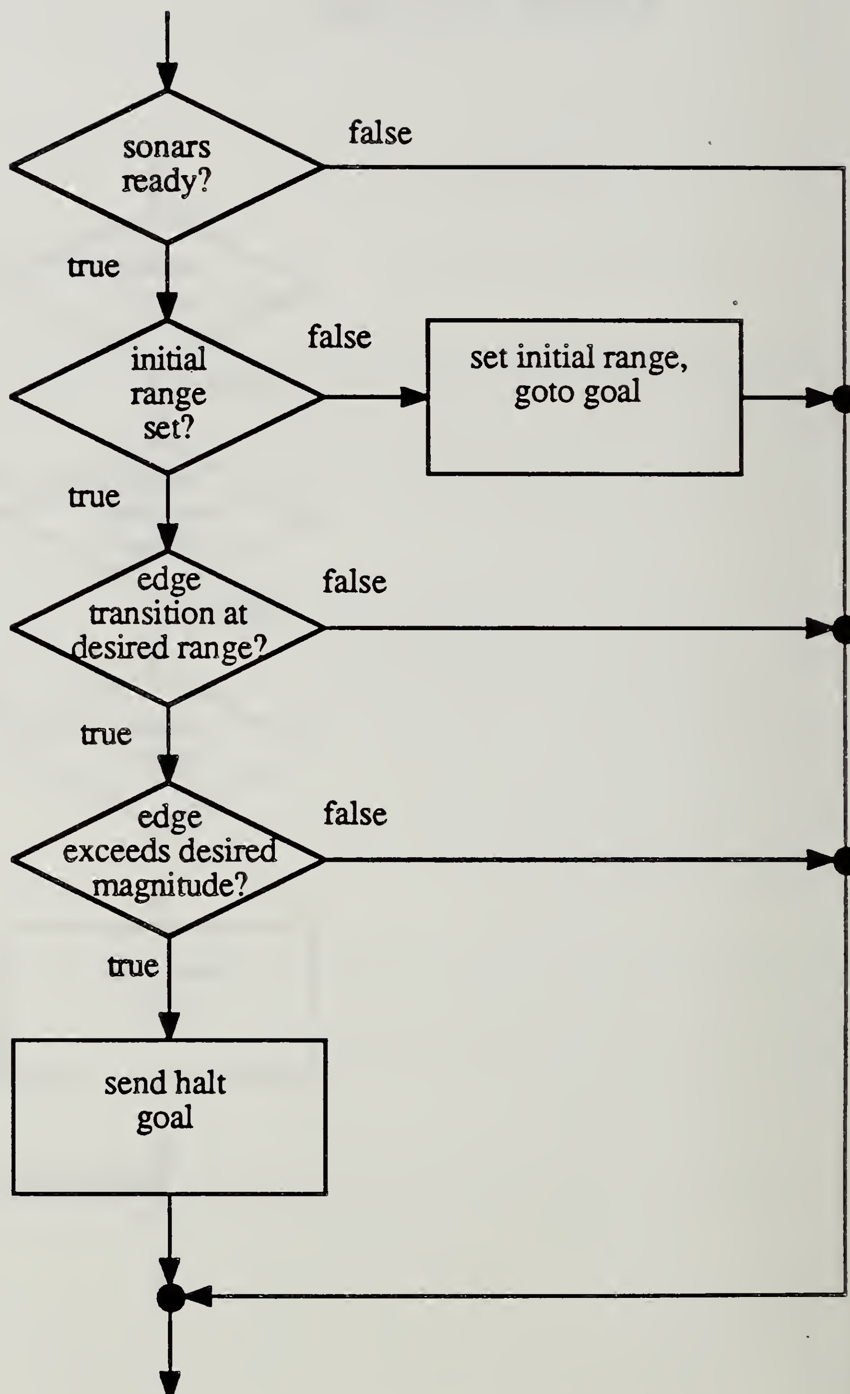
5.9.2. DISPLAY-PROXIMITY-SWITCHES
Input:
Output: **switches**

This routine calls **SWITCH-READ** to update the status of the proximity switches. The value is printed if there is a change in the status of the switches. The routine runs in a loop until it is aborted from the RSL level by typing **ABT PATH**. Like **DISPLAY-SONAR-SELECTION** (section 5.9.1.), this routine should not be compiled into the background task.

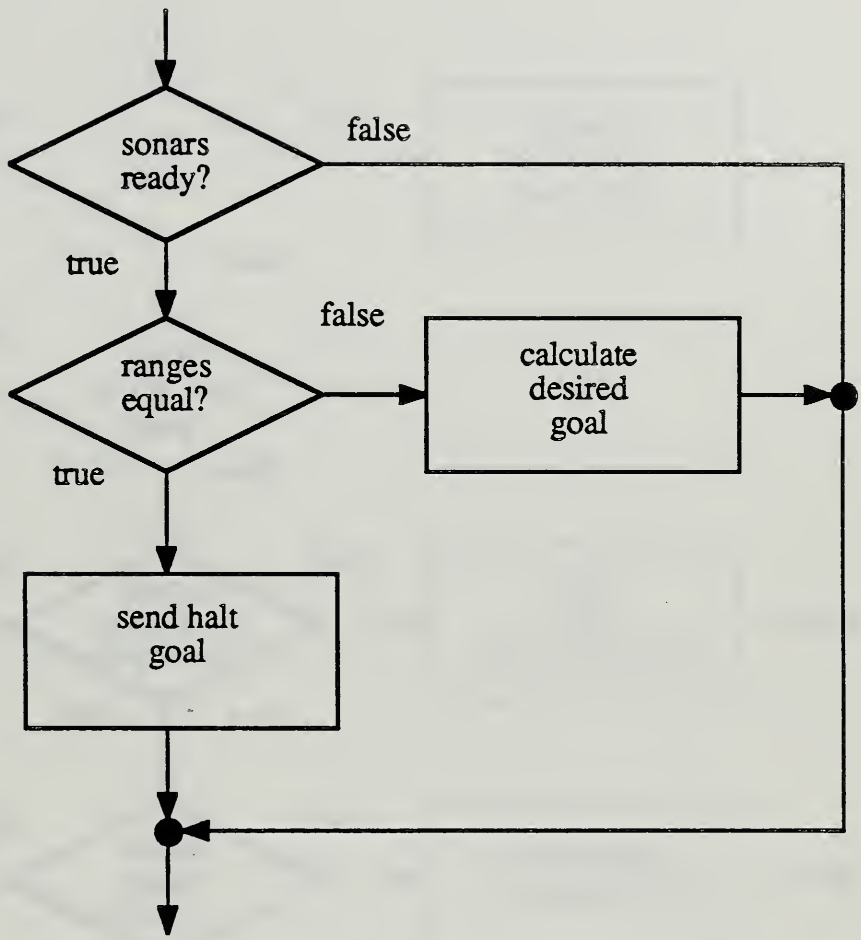
RANGE Command



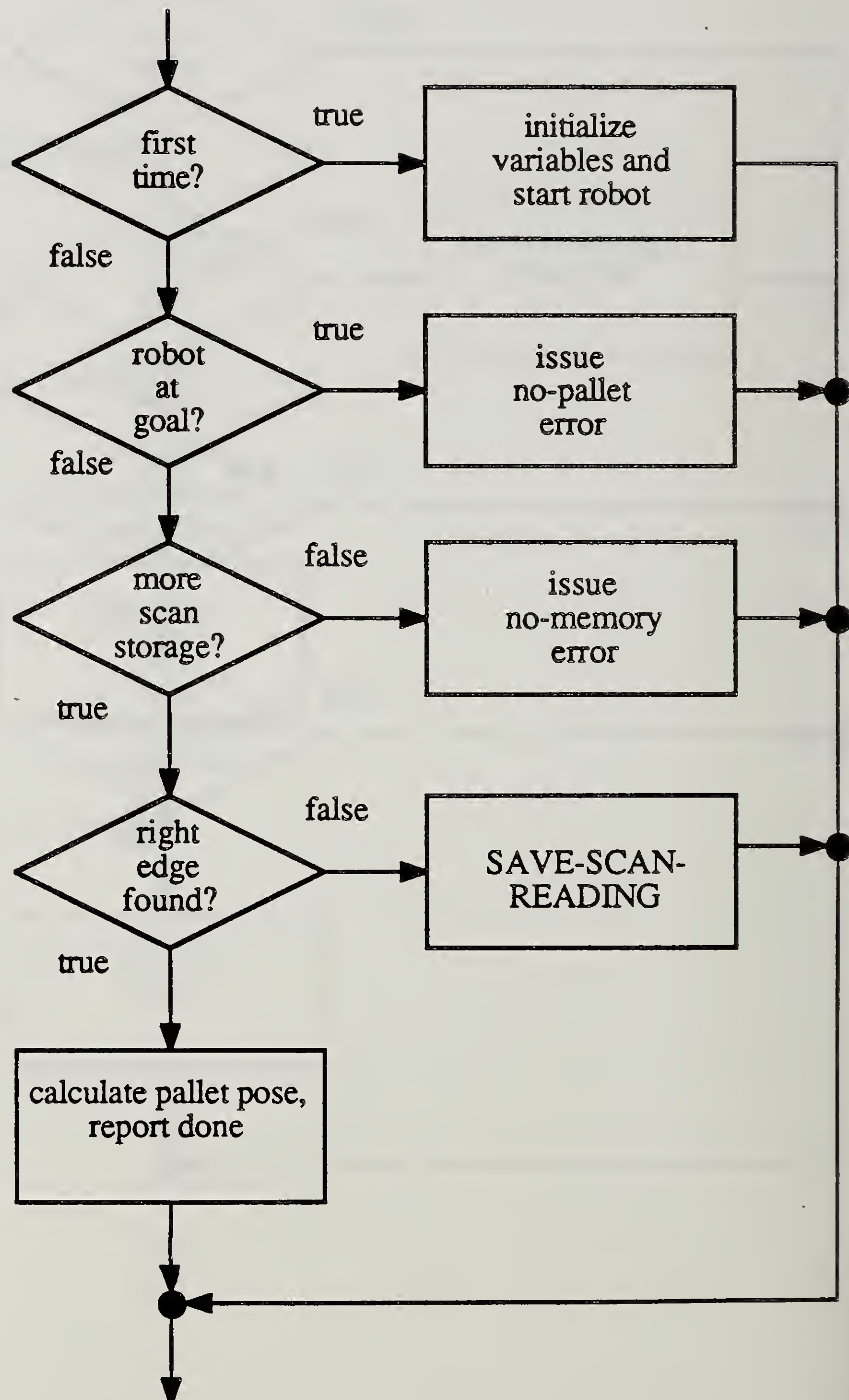
EDGE Command



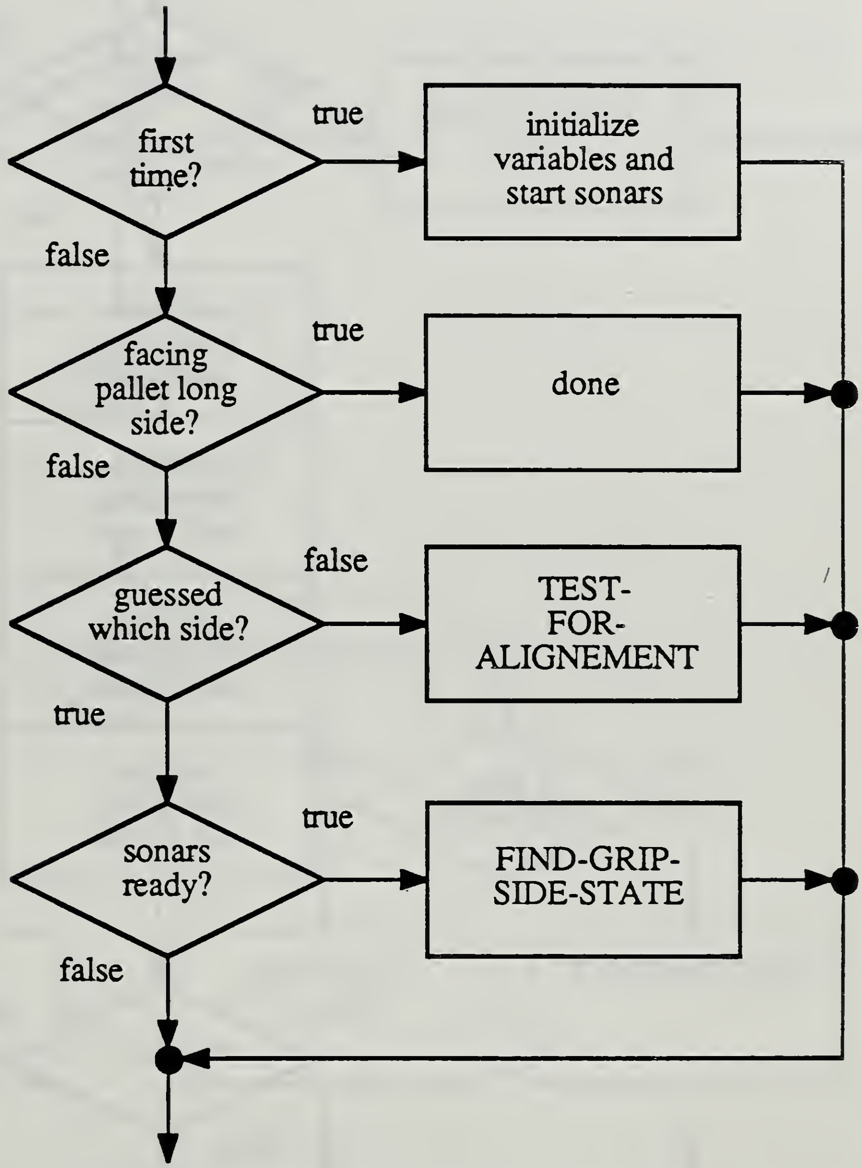
EQUATE Command



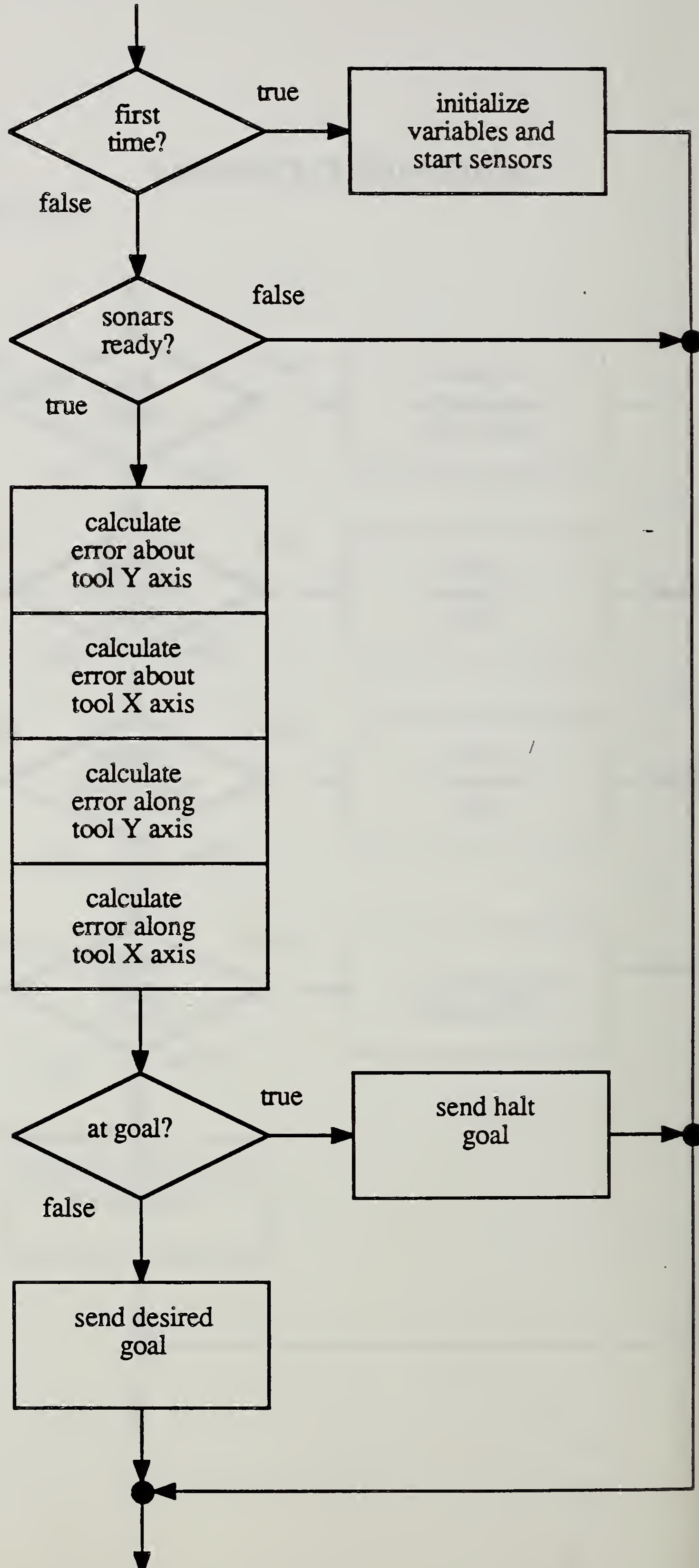
SCAN Command
SCAN-SONAR Routine



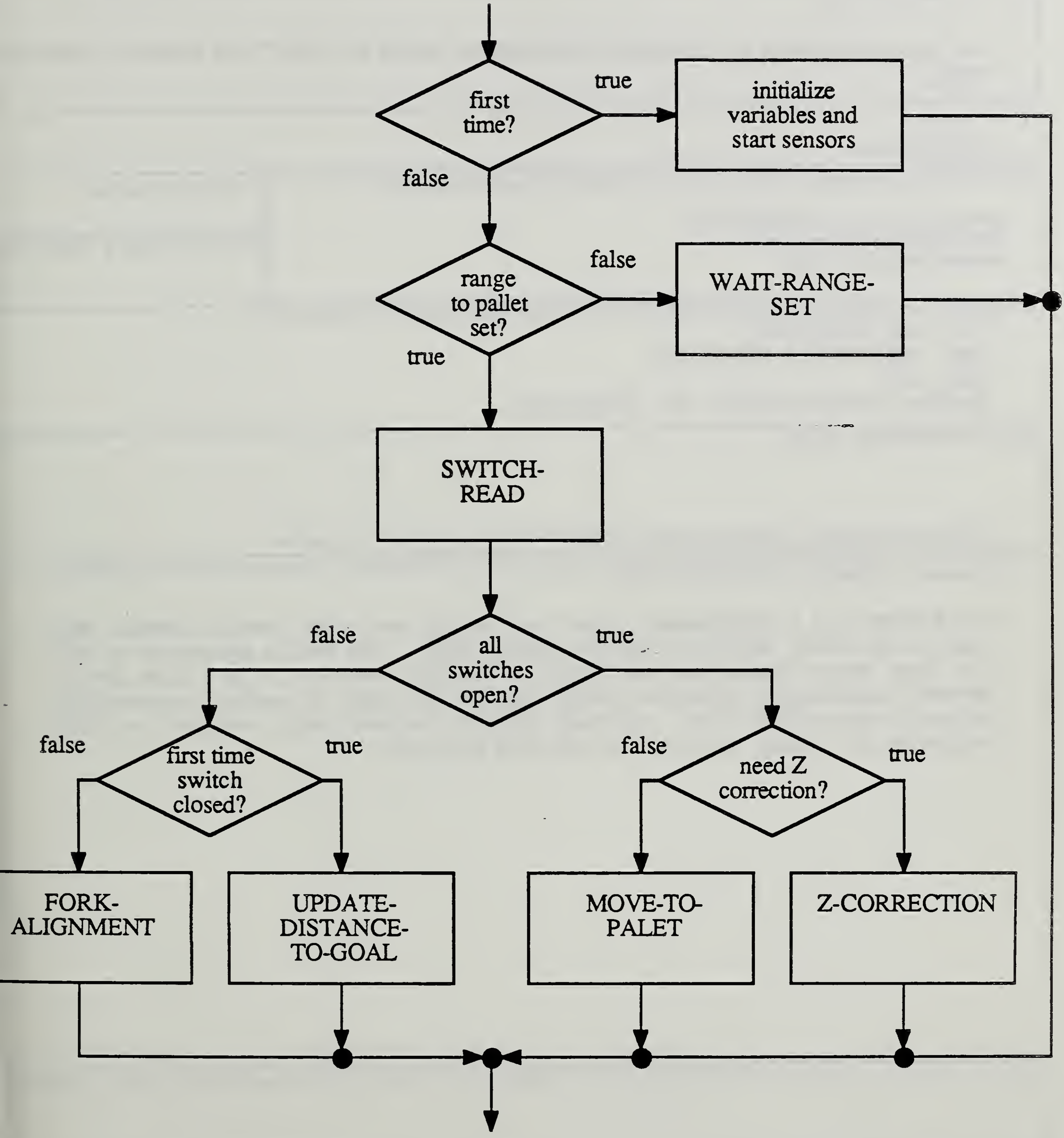
ALIGN-GRIP Command



APPROACH-PALLET Command



PICKUP-PALLET Command



NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 87-3624	2. Performing Organ. Report No.	3. Publication Date OCTOBER 1987

4. TITLE AND SUBTITLE

The National Bureau of Standards Programmers Guide for the Field Material Handling Robot.

5. AUTHOR(S)

Sandor Szabo

6. PERFORMING ORGANIZATION (*If joint or other than NBS, see instructions*)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (*Street, City, State, ZIP*)

U.S. Army Laboratory
Human Engineering Laboratory
Bldg. 459
Aberdeen Proving Ground, MD 21005-5001

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (*A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here*)

The document is a programmers guide for the NBS Real-Time Control System (RCS) used in the Field Material Handling Robot (FMR). The FMR is sponsored by the U.S. Army Human Engineering Laboratory. The RCS (version 2) is a high level, sensory interactive controller which enables the robot to perform automaticly as a stationary fork lift. The Robot Sensor Language (RSL) provides the RCS a mechanism for sensor integration and task planning.

12. KEY WORDS (*Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons*)

13. AVAILABILITY

[X] Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
[X] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

58

15. Price

$13.95